

Intel® Developer Cloud for the Edge Container Playground

Contents

Chapter 1: Intel® Developer Cloud for the Edge Container Playground

What's New?	4
Concepts	4
Marketplace (QuickStart)	6
Test Containers	10
Import containers from registries	11
Configure imported containers	12
Select Hardware and Launch Containers	14
Test with helm-charts	14
Test with docker-compose	18
View status, output and performance	20
Import Code and Build Containers	22
Build with dockerfiles	23
Build from application source without dockerfiles	24
Develop Containers in JupyterLab	27
Run OpenVINO™ notebooks	28
Access OpenVINO™ utilities	31
Build containers from terminal	31
Enable Cloud Storage	33
Available Hardware	41
Frequently Asked Questions	43
Known Limitations	45
About containers, helm-charts and docker-compose	45
Notices and Disclaimers	46

Intel® Developer Cloud for the Edge Container Playground

1



This guide is designed to help you get familiar with all the capabilities of the Intel® Developer Cloud for the Edge Container Playground. Use the **Notes** at the end of each section to try out working examples of each capability.

Introduction

The Intel® Developer Cloud for the Edge Container Playground extension enables you to seamlessly develop, build and test cloud-native container applications on various target deployment hardware.

- Intel® Xeon® and Intel® Core™ processors
- Intel® Processor Graphics including Intel® Iris® Xe

The Container Playground can be used to securely test, optimize and enhance your applications before deploying your solution on a cloud service provider or locally on edge devices.

Marketplace

To get started, try out the full-stack reference implementations, configurable AI Solutions, notebooks to learn about OpenVINO™ API or use samples as building blocks for your custom solutions.

Develop

Use the JupyterLab environment to download, convert or optimize pre-trained models from supported deep learning frameworks with the Intel® Distribution of OpenVINO™ Toolkit.

Build

Import your source code hosted in a git repository and build your container images directly from the application source or Dockerfiles.

Test

Benchmark your existing container images imported from container registries. You can also test your multi-container solutions with docker-compose files and helm charts.

Get an overview using the **Quick tour** option.



NOTE

- Current release supports **CPUs** and **integrated GPUs**.
-

What's New?

2022.3.0

- Support for [cloud connector functionality](#), which provides file sharing capability between Container Playground and Cloud Storage.
- Bug fixes.

2022.2.1

- Support for [hardware utilization](#) (CPU, GPU, memory, temperature metrics) for launched projects.
- Bug fixes.

2022.2.0

- Support for launching root containers without privileged mode.
- Access the [container terminal](#) for easy debugging.
- Support for editing and launching previously run projects directly from the Dashboard.
- Launch reconfigurable solutions from Intel® Developer Cloud for the Edge [Marketplace](#).
- Support to launch a project on up to [three platforms simultaneously](#) at one go.
- New ONNXRuntime OpenVINO Execution Provider samples from Intel® Developer Cloud for the Edge [Marketplace](#).
- New 12th and 11th Generation [platform configurations](#) now available.

2022.1.0

- Deployments with [helm-charts](#) and [docker-compose](#) are officially supported.
- Support for browser accessible URLs for container workloads.
- Access to a collection of [OpenVINO notebooks](#) from Intel® Developer Cloud for the Edge Container Playground Marketplace.
- Support for [customizing Marketplace sample applications](#) from source code with View/Edit Code option.
- New [filesystem experience](#).
- Importing containers from [AWS and Azure Private registries](#) is supported.
- Support for [mounting storage path](#) for runtime access in containers.
- [Quick Tour](#) capability to get familiar with key Container Playground components.
- New [platform configurations](#).

Concepts

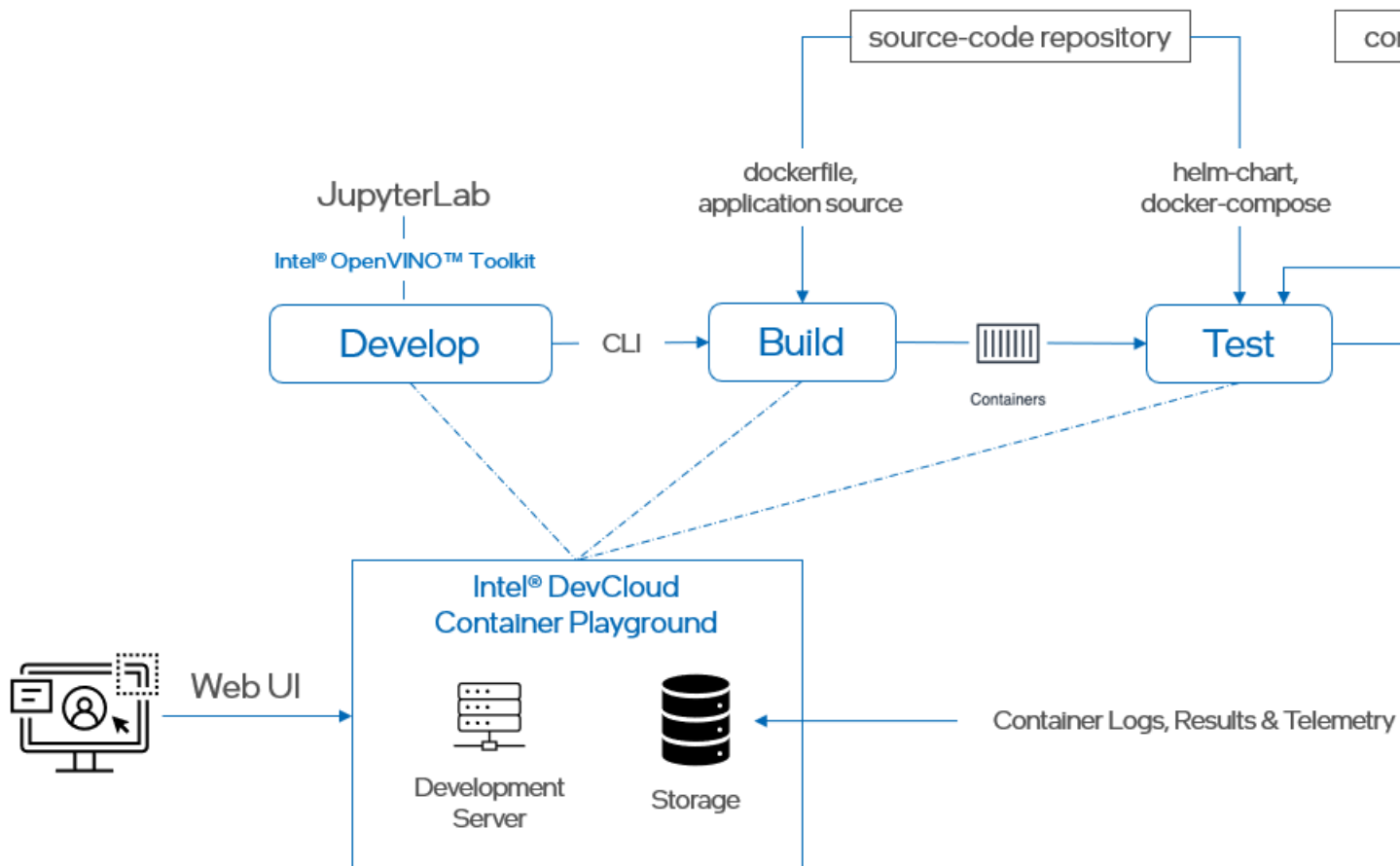
Use the Container Playground components to **develop**, **build** and **test** your applications on Intel® hardware directly from your browser without any setup required on your local machine.

Get an overview using the **Quick tour** option.



Component	Description
Resources	Import assets like container images, dockerfiles, helm charts, docker-compose files, and application source code.
Projects	Associate imported resources with one or more projects to configure and run.
My Library	Manage your configured projects, imported resources, and storage filesystem.
Dashboard	Start, monitor, view performance information, stop and remove your active projects.
Marketplace	Pre-built container catalog to try out or use as building blocks for your projects.
CLI	JupyterLab IDE with buildah and OpenVINO™ installed for interactive development and customizing sample applications.

How It Works?



NOTE

- Your account will be active for **120 days** after sign-up.
 - A single logged in session can be used for up to **4 hours**.
-

Testing environment

The container-playground portal supports multiple mechanisms to onboard your containerized applications.

- [Import containers from registries](#)
- [Test with docker-compose](#) files or [Test with helm-charts](#) hosted in Git repositories.
- [Build with dockerfiles](#) or [Build from application source without dockerfiles](#) hosted in Git repositories.

Above **imported resources** must be associated with a **project** to configure and launch on [available Intel® hardware](#). Upon launch, the target hardware is allocated to your user account **exclusively** until the execution completes or a maximum time limit of **30mins** is reached. You also have access to the container logs, performance information and a filesystem that can be mounted to retrieve data from the containers.

NOTE

- You can launch up to **3 different projects** on target platforms simultaneously.
 - Use the **private registry** associated with your account to store up to **15 containers** with a total storage limit of **20GB**
 - Your containers can be built with a maximum size of **10GB**
-

JupyterLab environment

The JupyterLab environment can be used for **development** tasks such as:

- [onboarding models to OpenVINO™ IR format](#)
- [getting familiar with the OpenVINO™ inference engine api](#)
- [editing marketplace sample applications and building containers from the terminal](#).

Your workspace includes:

- Intel® Xeon® processor with **2 vCPUs** and **4 GB RAM**
- Access to **2 GB** of private persistent filesystem storage associated with your account expandable up to 5GB
- OpenVINO™ Toolkit utilities accessible in the terminal
- Pre-installed dependencies to run OpenVINO™ notebooks
- Essential utilities such as Python, git, curl, wget and Java
- Buildah to facilitate building your containers from dockerfiles

NOTE

- Your active Jupyterlab instance can be used for up to **2 hours**.
 - For performance benchmarking tasks, see [Test Containers](#).
-

Marketplace (QuickStart)

The Intel® Developer Cloud for the Edge Marketplace hosts a growing list of pre-built container catalog targeting a variety of sample applications and reference implementations to try out or use as building blocks for your workloads.

Sample Applications	Showcase common use-cases like People Counter.
Reference Implementations	Full-stack multi-container workloads such as Intelligent Traffic Management.
Configurable AI Solutions	Multi-camera workloads such as Smart Retail Analytics.
OpenVINO™ Notebooks	Collection of Jupyter notebooks for learning and experimenting with the Intel® Distribution of OpenVINO™ Toolkit.

Use the top navigation menu to access the Intel® Developer Cloud for the Edge **Marketplace**.

NOTE

- Click on **Details** to preview the details such as the software stack, task (e.g. classification), container images and models.
- Use the **View/Edit Code** option to open the JupyterLab environment to customize and rebuild a sample application. See [building containers from the terminal](#) for more info.

Sample Applications

Launch the People Counter System or use **See All** to view additional sample applications.

- On the project information page, use **Launch** again from the bottom-right.
- Launch** the application on a target device by selecting a *target platform family* and a processor from the list of *platform configurations*.
- In your **Dashboard** view, once the project status updates to **Running** or **Completed**, expand the project row to **View Logs**.

The screenshot shows the Intel Developer Cloud interface. At the top, there are three buttons: 'Launch' (with a rocket icon), 'Remove' (with a minus icon), and 'History' (with a clock icon). Below these is a table with the following columns: 'Project', 'Target', 'Status', and 'Execution Time'.

Project	Target	Status	Execution Time
people-counter-syst...	Core i7-12700	Completed	29 sec

Below the table, there is a dropdown menu for the 'people-counter-syst...' project. The dropdown is open, showing a list of deployment names and their status. The first entry is '0-people-counter-2021-4-21659572679533-vmx...' with a status of 'Completed'.

Deployment Name	Label	Status
0-people-counter-2021-4-21659572679533-vmx...		Completed




From the Filesystem tab of the MyLibrary view, navigate inside the project
`output_people_counter_latest > results > FP16` to preview or download the result files.

Reference Implementations

Scroll down and **Launch** the Intelligent Traffic Management application.

1. On the project information page, use **Launch** again from the bottom-right.
2. **Launch** the application on a target device by selecting a *target platform family* and a processor from the list of *platform configurations*.
3. In your **Dashboard** view, once the project status updates from **Queued** to **Running** use the **Outputs** option to access web-service containers such as grafana dashboard.

Q

 Launch
 Remove
 History

Project	Target	Status	Execution Time	Last Executed	Process
> intelligent-traffic-...	<u>Core i7-11700</u>	● Running	1 min, 19 sec	Mar 30, 2022	N

4. Access the ITM dashboard using the *Manage* option from the *Dashboards* menu on the left.

After trying out the application use **Stop** or **Remove** to free-up your assigned target device. By default, your project will be terminated after **30mins**.



Configurable AI Solutions

Scroll down and **Configure & Launch** the Multi Camera Social Distancing solution.

1. On the Reconfigurable Solutions page *select* the use-case.
2. Use the arrow on the right for each use-case to further configure **input data source** and **input model**.
 - **Sample Videos:** Use sample videos provided by Intel® Developer Cloud for the Edge.
 - **Devcloud File path:** Specify any of your uploaded videos present on the Intel® Developer Cloud for the Edge Filesystem.
 - **External RTSP url:** Use stream provided by an external RTSP server.

The number of camera streams can be configured in the range of 1 to 16.

3. **Launch** the application on a target device by selecting a *target platform family* and a processor from the list of *platform configurations*.

4. In your **Dashboard** view, once the project status updates to **Running** use the **Outputs** option to access the dashboard.

The screenshot shows the 'Reconfigurable Solutions' interface for the 'multi-camera-social-distancing' project. The main configuration panel on the left is titled 'social-distancing' and includes the following sections:

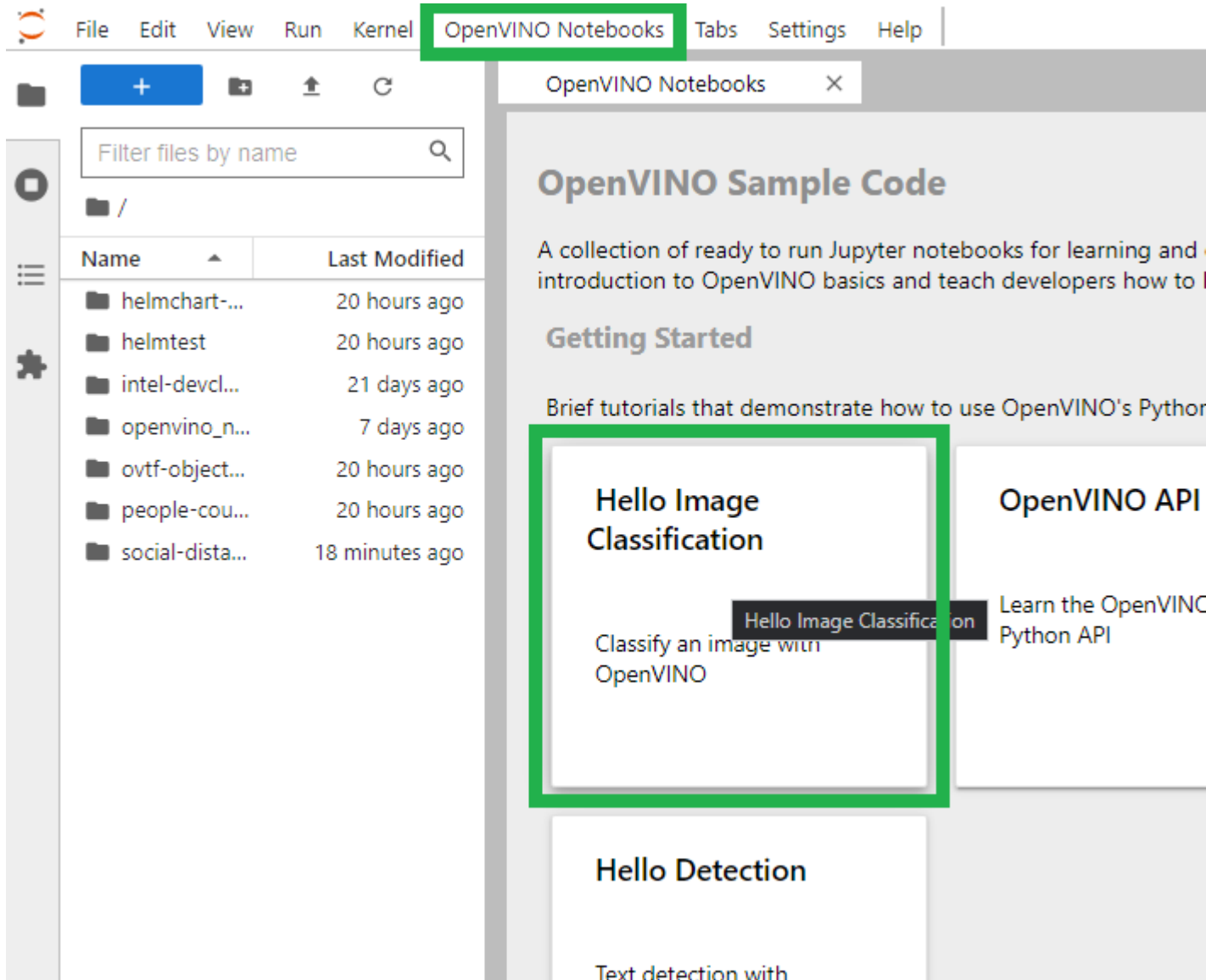
- Configure**: A section with a cursor icon.
- Number of streams selected**: A dropdown menu set to '2'.
- Data source**: A dropdown menu set to 'Sample video'.
- File path**: A text input field.
- Select Model**: A section with a dropdown menu.
- Person Detection**: A dropdown menu set to 'person-detection-0202'.

On the right side of the interface, there is a video feed showing a crowd of people with green bounding boxes around them. Overlaid text on the video reads 'use-case=social-distancing' and 'violations=7'. Below the video feed, there is a 'Performance Data' section with a circular gauge showing the 'Number of Input streams' as '2'.

OpenVINO™ Notebooks

Scroll down and use **View/Edit Code** on OpenVINO Notebooks to open the JupyterLab interface.

1. From the menu, select to open a notebook in a new tab inside the JupyterLab interface.



2. Make sure a **valid OpenVINO kernel** is selected and run the cells to start experimenting and learning. For more info, refer to [running openvino notebooks in JupyterLab IDE](#).

NOTE

- Only the Notebooks listed on the menu are officially supported.
- Use the notebooks to learn the api and not for benchmarking purposes, as the notebooks are running in a development environment.

Test Containers

If you already have containerized applications ready to test on Intel Platforms, this section shows you how to import, configure, launch and analyze your containers with just a few clicks.

You can import one or more containers and provide container runtime configurations or directly test using docker-compose files or helm-charts. Use the examples provided to prepare your containers to display performance information in your Intel® Developer Cloud for the Edge dashboard, save outputs to your Intel® Developer Cloud for the Edge filesystem, or simply display logs.

- [Import containers from registries](#)
 - [Navigate to Container from Registry](#)
 - [Import from external registry](#)
 - [Import from internal registry](#)
- [Configure imported containers](#)
 - [Navigate to Configure](#)
 - [Container Configurations](#)
 - [Container-to-Container Communication](#)
- [Select Hardware and Launch Containers](#)
 - [Access the Launch Option](#)
 - [Configure Target Hardware](#)
- [Test with helm-charts](#)
 - [Navigate to Helm Chart](#)
 - [Configure Import](#)
- [Test with docker-compose](#)
 - [Navigate to Docker Compose](#)
 - [Configure Import](#)
- [View status, output and performance](#)
 - [Monitor Status](#)
 - [Container Logs](#)
 - [Performance Dashboard](#)
 - [Data in Filesystem](#)
 - [Web Service URLs](#)
 - [Container Terminal Access](#)

Import containers from registries

You can bring your existing linux containers from [Docker Hub](#), [Red Hat Quay.io](#), [Amazon ECR](#), [Azure](#), [Google](#) container registries. To reuse containers without fetching them everytime or associate the same container across multiple projects, a private registry is also provided for your Intel® Developer Cloud for the Edge account.

Navigate to Container from Registry

Use your **Dashboard** view to access the **+Import Resources** option and select **Container from Registry**.

First, select the target project from the drop down to associate containers to be imported. If you don't have a project already, use the **+Create New Project** option to continue importing from an internal or external registry.

Import from external registry

Enter the required details for importing from a *public* container registry.

- **Container Url:** Repository URL with a tag, e.g. for docker hub `docker.io/user/repo:tag`
- **Add this to internal registry:** To reimport the container quickly in another project.
- **Container Name:** Unique identifier to recognize across other Container Playground views.

To import from a *private* registry, ensure **Private** is checked and enter the additional details.

- **Registry Server URL:** e.g. for docker hub private repository `https://docker.io`

- **Username:** username used to log in to your registry
- **Password:** your private registry password

NOTE

- The Url for AWS will be in the form of `<aws_account_id>.dkr.ecr.<region>.amazonaws.com/<container_name>:<tag>` and `<container_name>.azurecr.io` for Azure.
 - To import from AWS ECR private registry, first make sure your IAM user is assigned [AmazonEC2ContainerRegistryReadOnly](#) and above policy. Retrieve the *Access Key ID*, *Secret Access Key* and use the `aws configure` command using the aws cli to login locally. Finally, retrieve the **AWS ECR token** using the command `aws ecr get-login-password --region <region>` which must be entered in the **password** field.
 - To import from Azure ACR private registry, setup and retrieve the **username** and **password** from *Settings > Access Keys* from the Azure portal.
-

Import from internal registry

Use this option to select a container previously imported with the **Add this to internal registry** option. Check all the containers from list of containers you would like to associate with this project.



Click **import** on the bottom right to view the list of containers associated with the current project. You can also continue importing additional containers with **+Add new Containers** option.

See [Configure imported containers](#) and [Select Hardware and Launch Containers](#) to launch.

NOTE Try out the above flow with [docker hello-world](#) container from dockerhub registry:

- Project Name: `helloworld-project`
- (Import From External Registry) Container Url: `docker.io/hello-world`
- Container Name: `helloworld-container`

Use the **View Logs** option to see the console output of the container.

Configure imported containers

Navigate to Configure

While importing containers from registry use the **Configure** option. You can also configure at a future time by navigating to **Projects** under **My Library** view and clicking on **Configure** icon under **Actions** next to your project.

Projects Resources Filesystem			
Project Name	Creation Date	Assigned Resources	Description
analytics-bench...	Oct 24, 2021	1 Container(s)	

[Previous](#)

Container Configurations

For each container in your project, you can individually configure:

- **Port:** Expose ports for multi-container communication. Use **+Add** to apply.
- **Enable Routes (Toggle):** First toggle-on and then add a port above to ensure a **checkmark** is shown to enable browser accessible URL. (maximum 1)



- **Labels:** Create labels to keep a track of custom configurations of an imported container across projects.
- **Entry Point:** Use to override the default entrypoint.
- **Output Mount Point:** Retrieve result data from the container to preview later in your filesystem.
- **Volume Binder > Filesystem Path:** Path on the Intel® Developer Cloud for the Edge filesystem (maximum 5)
- **Volume Binder > Input Mount Point:** Path inside the container (maximum 5)
- **Dependency:** In multi-container scenarios select the other dependent containers that must be running before this container begins running.
- **Configuration Parameters:** Pass runtime container *environment variables* for dynamic container configurations e.g., `-e VAR1=VALUE1 -e VAR2=VALUE2`. For alternate entrypoint parameters, also referred to as *command args*, use docker-compose or helm-charts instead.

NOTE

- Make sure to check for valid configurations such as mount points to avoid unexpected launch failures.
- Valid paths cannot contain spaces.

You can also reorder the sequence of running containers associated with a project by dragging the specific container around.



Use **My Library** to launch from your projects tab, see [Select Hardware and Launch Containers](#).

Container-to-Container Communication

To enable container communication inside a project, use **Container Name** and replace **:** with **-** as the address to connect to. e.g., for resource name `test-server:latest`, provide `test-server-latest` in the client container program to establish a connection

You can also enable container communication across target platforms using the above mechanism by launching multiple projects.

Select Hardware and Launch Containers

Access the Launch Option

Navigate to your **Projects** tab under the **My Library** view and use the **rocketlaunch** icon next to your project name to proceed to the selection of target hardware.

Configure Target Hardware

Select a *target platform family* and a processor from the list of *platform configurations*. You can also multi-select up to 3 processors and launch simultaneously.



Your active project can be monitored from the **Dashboard** view. The status of your project can be in **Queued**, **Deploying**, **Running**, **Completed**, **Failed** or **Error** states.

See the [View status, output and performance](#) section to further analyze your project.

NOTE If your program running inside the container is utilizing the GPU, and you target a processor that supports an integrated GPU the Intel® Developer Cloud for the Edge portal will auto-enable GPU access. No other code changes are required.

To dynamically benchmark your application without changing your code and rebuilding your container, you can optionally leverage the environment variables passed in runtime to switch between CPU or GPU before launching, see [Configure imported containers](#).

Test with helm-charts

You can import helm-charts either from existing repositories like [bitnami](#) and [artifactory.io](#) or a source code repository. For code repositories the portal will look for an **index.yaml** file that references one or more archived charts ending with **.tgz**.

You can create packaged charts from a chart directory with `helm package PATH_TO_CHART_DIR` and generate the index file with `helm repo index` ahead of time.

Navigate to Helm Chart

Use your **Dashboard** view to access the **+Import Resources** option and select **Helm Chart**.

Configure Import

1. Enter the configuration of the source code repository.

- **Repo URL:**

- For repositories like bitnami use paths generally used with the `helm repo add` command e.g. `https://charts.bitnami.com/bitnami`.
- For source code repositories enter the raw github path format. To retrieve the raw github filepath click open the index.yaml file in your browser and select to open **Raw** file contents. Remove the trailing filename to only include the repository name and branch e.g. `https://raw.githubusercontent.com/intel/DevCloudContent-helm/main`

- **Private:** Check to import from private repository

- **Secret:** For private repos, enter your git account **Username** and the **Git Token**.

To generate a secret, navigate to your github account `Settings > Developer Settings > Personal Access Tokens > Generate New Token` and ensure `repo` access is enabled. Your one-time token will be generated in the next page.

Hit **Verify** on the bottom-right.

2. Enter a unique **Repo Name** to identify your docker-compose resource in the Intel® Developer Cloud for the Edge portal. You will be presented a list of helm-charts referenced in your index.yml.

Select the single helm-chart of interest and proceed to the next page with **Submit**.

Repo URL

Specify a URL for a HELM Chart in a repository

`https://raw.githubusercontent.com/DevcloudContent/helm-import/reference-impl`

Repo Name*

`helm-import`

Repo Name must consist of lower case letters, numbers & hyphens with a minimum of two and a max of 60 characters. It must start with a letter and end with a letter or a number.

☐ Private

If selected, you'll be prompted to enter credentials to a private registry

Chart Name*

`social-distancing-helm`

|

`wireless-ready-itm-helm`

`social-distancing-helm`

3. The container Playground portal will show the modified changes of your original helm-chart required to run successfully inside your Intel® Developer Cloud for the Edge account.

You must select an existing project or create one with **+Create New Project** option to associate your imported resource.

HELM Chart

Resource Name	Status	Version
helm-import-social-distancing-helm	● Completed	

Values.Yaml

The Helm chart has been imported and is ready to run without any modifications to values.yaml

Original Chart

```

986     image: "default-route-openshift-image-registry.apps.cfa.devclo
987     imagePullPolicy: "Always"
988     name: "mc-ssr"
989     volumeMounts:
990     - mountPath: "/app/camera_config.json"
991       name: "mc-ssr-config"
992       readOnly: true
993       subPath: "camera_config.json"
994
995     volumes:
996     - configMap:
997       name: "mc-ssr-config"
998       name: "mc-ssr-config"

```

Modified Chart

```

994     image: "default-route-openshift-image-registry.apps.cfa.devclo
995     imagePullPolicy: "Always"
996     name: "mc-ssr"
997     volumeMounts:
998     - mountPath: "/app/camera_config.json"
999       name: "mc-ssr-config"
1000       readOnly: true
1001       subPath: "camera_config.json"
1002+   nodeSelector:
1003+     edgenode: "${NODE_LABEL}"
1004+   serviceAccountName: "ojsasda"
1005     volumes:
1006     - configMap:
1007       name: "mc-ssr-config"
1008       name: "mc-ssr-config"

```

4. After assigning a project, use the **My Library** on the bottom to return to the My-Library view.

To launch your project, see [Select Hardware and Launch Containers](#).

NOTE Try out the above flow with the pre-configured git repository containing an example index.yaml:

- Repo URL: <https://raw.githubusercontent.com/intel/DevCloudContent-helm/main>
- Repo Name: helm-import
- Select social-distancing-helm from the Chart Name drop down list.
- Assign the resource to a new project helmchart-example on the changes comparison page.

Four containers will be in running state. Use the [dashboard view](#) to see the logs of social-distancing mc-ssr container that utilizes the dstreamer api to perform inference on a video, mqtt (mosquitto) broker container and results written to influxdb container.

For the grafana container, open the dashboard URL. With the menu on the left, navigate to Dashboards > Manage > Multi Camera Covid-19 Solution to see the violations.

If you face issues, try deleting the project from My Library before launching.

Test with docker-compose

You easily can run multi-container applications on various Intel Platforms by importing docker compose YAML files hosted on external Github repositories. Common docker-compose configurations such as *depends_on*, *ports* and *build* are also supported.

Make sure to place a single docker-compose file in the root of the repo as the container-playground portal will look for the first available `docker-compose.yml` in your git repo.

Navigate to Docker Compose

Use your **Dashboard** view to access the **+Import Resources** option and select **Docker Compose**.

Configure Import

1. Enter the configuration of the source code repository.
 - **GIT Repo URL:** Enter the root browser url of your source code repository
 - **Private:** Check to import from private repository
 - **Secret:** For private repos, select an existing secret or create one using the **+Create New Secret** option. Enter your **Secret Name**, git account **Username** and the **Git Token**.

To generate a secret, navigate to your github account `Settings > Developer Settings > Personal Access Tokens > Generate New Token` and ensure `repo` access is enabled. Your one-time token will be generated in the next page.

Hit **Verify** on the bottom-right.

2. Enter a unique **Resource Name** to indentify your docker-compose resource in the Intel® Developer Cloud for the Edge portal and proceed to the next page with **Submit**.
3. The container Playground portal will show the modified changes of your original docker-compose file required to run successfully inside your Intel® Developer Cloud for the Edge account.

You must select an existing project or create one with **+Create New Project** option to associate your imported resource.

Resource Name

Status

openvino-modelserver

● Completed

docker-compose.Yaml

The import docker compose ready to run on Devcloud with some modifications to docker-compose.yaml

Original Chart

```

1 ---
2 apiVersion: "template.openshift.io/v1"
3 kind: "Template"
4 metadata:
5   creationTimestamp: "2021-10-24T20:12:39Z"
6   managedFields:
7     - apiVersion: "template.openshift.io/v1"
8       fieldsType: "FieldsV1"
9       fieldsV1:
10         f:objects: {}
11   manager: "okhttp"
12   operation: "Update"
13   time: "2021-10-24T20:12:39Z"
14   name: "openvino-modelserver-old"
15   namespace: "ojasdsawant21-intel"
16   resourceVersion: "476206162"
17   uid: "2de9e425-fd0f-433b-b704-f9cc92a237ee"
18 objects:
19 - apiVersion: "v1"
20   kind: "Service"
21   metadata:
22     labels:
23       io.kompose.service: "ovms-client"

```

Modified Chart

```

1 ---
2 apiVersion: "template.openshift.io/v1"
3 kind: "Template"
4 metadata:
5   creationTimestamp: "2021-10-24T20:12:39Z"
6   managedFields:
7     - apiVersion: "template.openshift.io/v1"
8       fieldsType: "FieldsV1"
9       fieldsV1:
10         f:objects: {}
11+      f:parameters: {}
12   manager: "okhttp"
13   operation: "Update"
14   time: "2021-10-24T20:12:39Z"
15+   name: "openvino-modelserver"
16   namespace: "ojasdsawant21-intel"
17+   resourceVersion: "476206161"
18+   uid: "de0d3fc6-8e78-4425-b0b5-c0fa"
19 objects:
20 - apiVersion: "v1"
21   kind: "Service"
22   metadata:
23     labels:
24       io.kompose.service: "ovms-client"

```

- (Optional) Check the **Enable Routes** option to additionally configure browser accessible url for a service port specified in your docker-compose file.
- After assigning a project, use the **My Library** on the bottom to return to the My-Library view.

To launch your project, see [Select Hardware and Launch Containers](#).

NOTE Try out the above flow with the pre-configured git repository containing an example docker-compose file:

- GIT Repo URL: <https://github.com/intel/DevCloudContent-docker-compose>
- Resource Name: openvino-modelserver
- Assign the resource to a new project dockercompose-example on the changes comparison page.

Use the [dashboard view](#) to see log output of ovms-server container serving a detection model and ovms-client container performing multiple grpc inference requests. The ovms-server will be in running state until stopped and the ovms-client container will reach **completed** state.

View status, output and performance




Depending on the configuration and behavior of your container application, you can analyze your workload in multiple ways. You can use this section as a reference to prepare your container as well.



Monitor Status


The status of your project can be monitored with **Queued**, **Deploying**, **Failed** or **Error**. Depending upon the expected behaviour of your container, the status can be in **Running** state for up to **30mins** or **Completed** if the program completes execution.

Container Logs

From the **Dashboard** view of the Container Playground, expand your project run and click on the **Log Output** icon. Use the **Refresh Stream** button in the pop-up to refresh the latest console logs of your container application while in the **Running** or **Completed** state.

 Launch
  Remove
  History

Project	Target	Status	Execution Time	Last
<div>  people-counter-syst... </div>	Core i7-12700	 Completed	29 sec	At

Deployment Name	Label	Status
0-people-counter-2021-4-21659572679533-vmx...		 Completed

Performance Dashboard

Once your project reaches **Completed** status, expand your project and click on **Performance Metrics** to view detailed CPU, GPU, memory utilization and temperature metrics for the entire duration of your workload.



NOTE

- The greyed out **Performance metrics** icon will be enabled within few minutes of completion, without any additional project configuration.
- For projects launched with multiple containers on the same platform, the legend indicates **multi_container_workload** in resource sharing scenarios.
- If you come across **Telemetry Error** or **Telemetry Data expired** message, please [Request Support](#) (A product or service I already own or use > Search for a product or service by name > devcloud for the edge) .

You can also enable throughput (fps) and latency (ms) in the dashboard to quickly compare multiple runs on various Intel Platforms.

Launch Remove History				
Project	Target	Status	Execution Time	Last
> people-counter-system	Core i7-12700	● Completed	29 sec	Au

Your container must write out a `FP16/performance.txt` file in below format. You must also [Configure imported containers](#) with an **Output Mount Point** before launching.

```
Throughput: 104 FPS
Latency: 13457.780 ms
```

The Intel® Developer Cloud for the Edge portal recursively looks for the `performance.txt` file and `FP16` directory in your **Mount Point** or **Result Path** directories.

Data in Filesystem

You must [Configure imported containers](#) with a **Mount Point** before launching to enable storing data into the filesystem for your container. From the **Dashboard** view of the Container Playground, expand your project and click on the **FileSystem** icon next to the *Log Output* to open the FileSystem view from the **My Library** tab.



Web Service URLs

If your containers expose web-services such as grafana dashboard that can be accessed in a browser, you can **Expose Routes** while [configuring your imported containers](#).

Container Terminal Access

When the project is in **Running** state, you can gain access to the container terminal for easy debugging.

The screenshot displays the Intel Developer Cloud interface for the Edge Container Playground. At the top, there are three buttons: 'Launch' (with a rocket icon), 'Remove' (with a minus icon), and 'History' (with a clock icon). Below these is a table with columns: 'Project', 'Target', 'Status', and 'Execution Time'. The table contains one entry with 'Core i7-1185GRE' as the target, 'Running' status (indicated by a green dot), and '8 min, 57 sec' execution time. A modal window is open, showing a sub-table with columns: 'Deployment Name', 'Label', and 'Status'. This sub-table has one entry: '0-test-server-latest1659571360072-d8jfc' with a 'Running' status (green dot). At the bottom, there is a 'CP Terminal' section with a tab for 'test-server-latest' (which has a close button 'X'). The terminal window shows the text: 'Welcome to the container playground shell! " Let's get started with...' followed by a prompt '\$'.

Import Code and Build Containers

If you already have your application source code hosted on a git code repository, use this section to import and build containers with just a few clicks.

- Build with dockerfiles
 - [Navigate to Dockerfile](#)
 - [Configure Import](#)
 - [View build status](#)
- Build from application source without dockerfiles
 - [Navigate to Git Repo](#)
 - [Configure Import](#)
 - [Builder image](#)
 - [Builder with base image](#)
 - [View build status](#)

Build with dockerfiles

You can build containers from source code repositories containing **Dockerfiles**. If your repository contains multiple dockerfiles for each of your microservices, you can hand-pick the directory to invoke the dockerfile from and also specify the path with the name of the file with *.Dockerfile* extension.

Navigate to Dockerfile

Use your **Dashboard** view to access the **+Import Resources** option and select **Dockerfile**.

Configure Import

- Enter the details of your github repository containing dockerfiles.
 - GIT Repo Url:** Full browser URL pointing to the root of your repository.
 - Resource Name:** Unique name to indentify the built container across Container Playground views.

To import a **private repository**, check the *Private* box and select a secret from the drop down. If required, create a new secret with the **+Create New Secret** option.

- Secret Name:** For private repos, enter your git account **Username** and the **Git Token**.

To generate a secret, navigate to your github account Settings > Developer Settings > Personal Access Tokens > Generate New Token and ensure repo access is enabled. Your one-time token will be generated in the next page.

Hit **Verify** on the bottom-right.

- Enter additional configuration details to build your container:
 - Dockerfile Path:** Enter the name of the Dockerfile start from the root of the repository.
 - Git Branch:** Optional name of a specific branch e.g. `main`

To start building click **Build**.

View build status

Select the **My Library** option to continue monitoring the build status in the background from your resources view.

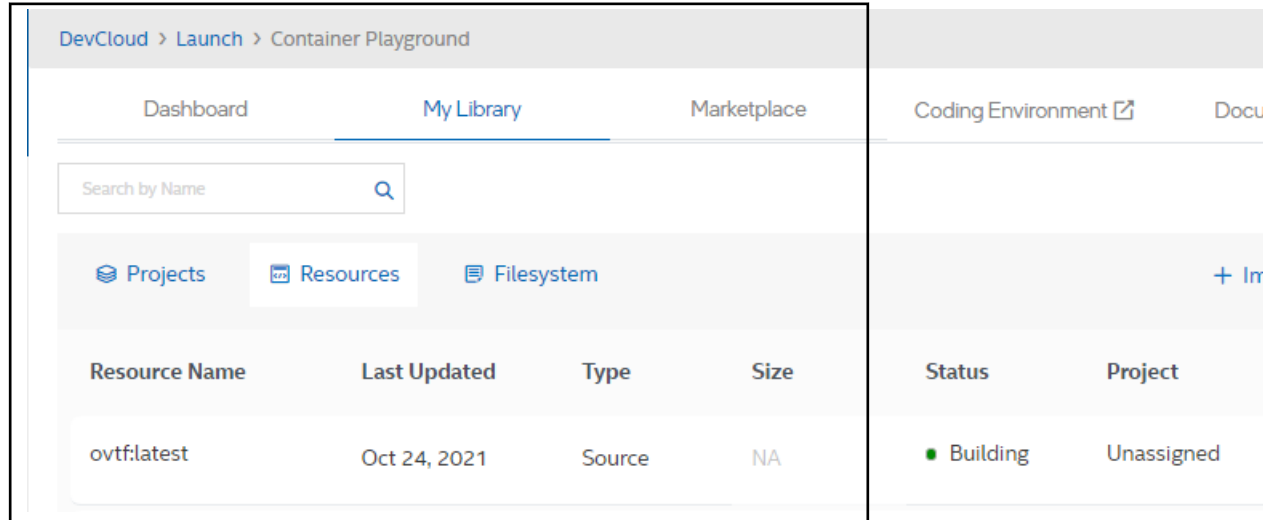
[Dashboard](#)
[My Library](#)
[Marketplace](#)
[Coding Environment](#)
[Documentation](#)

Import From Dockerfile

Image Name	Source	Status
ovtf:latest	https://github.com/DevcloudContent/dockerfile-import	● Building ⓘ

[My Library >](#)

From your **Resources** view, use the **Actions** option next to your *Resource Name* to view build logs, change build configuration or rebuild. Your status can show as **Building**, **Ready** or **Error**.



Once your resource status shows **Ready**, you must **Assign** the container to an existing project to additionally configure container runtime options and launch.

See [Configure imported containers](#) and [Select Hardware and Launch Containers](#) for more info.

Build from application source without dockerfiles

You can build containers directly from application source hosted in code repositories without a *dockerfile*. If your repository contains application source for multiple microservices, you can hand-pick the directory to package in your container.

Your application source can be in *Java*, *Python*, *PHP*, *NodeJS* and *Golang*. For applications requiring OpenVINO™ Toolkit dependencies, *CPP* and *Python* languages are supported.

Navigate to Git Repo

Use your **Dashboard** view to access the **+Import Resources** option and select **Git Repo**.

Configure Import

- Enter the details of your github repository containing your application source.
 - GIT Repo Url:** Full browser URL pointing to the root of your repository.
 - Resource Name:** Unique name to identify the built container
 - Secret Name:** For private repositories, select a secret from the drop down or use the **+Create New Secret** option to enter your git account **Username** and the **Git Token**.

To generate a secret, navigate to your github account `Settings > Developer Settings > Personal Access Tokens > Generate New Token` and ensure repo access is enabled. Your one-time token will be generated in the next page.

Hit **Verify** on the bottom-right.

- Enter additional configuration details to build your container.
 - Git branch:** Optional name of a specific branch e.g. `main`
 - Sub-directory:** Enter the path to the directory containing your application source.

Builder image

To build containers with no package dependencies, select the appropriate **programming language environment** for your container base image and an **Image Version**.

The Intel® Developer Cloud for the Edge portal will detect standard language specific source files from your code repository and build a ready-to-run container from your application source.

Programming Language	Source Files
Python	app.py for entrypoint and requirements.txt for installing any dependencies
Java	pom.xml
PHP	index.php and composer.json
NodeJS	server.js and package.json
Go	single .go extension file
NodeJS	server.js and package.json

Builder with base image

To build containers with pre-installed dependencies such as OpenVINO™ Toolkit, select the dependency and version from **Catalog Name** and **Version** drop-downs. Select the appropriate **programming language environment** and fill in the optional configurations.

- **Pre Build Script:** Optional setup shell script to download or extract packages
- **Working Dir:** Used as `mkdir [Working Dir]/build && cd [Working Dir]/build`
- **CMAKE Args:** Used as `cmake [CMAKE Args] ..`
- **CMAKE Build Args:** Used in current directory as `cmake --build . [CMAKE Build Args]`
- **Post Build Script:** Run any OpenVINO™ utilities like `downloader.py` or `convertor.py`
- **Entry Point:** Required to indicate shell script to run in container runtime

Catalog Name	Version	Information
OpenVINO™	2021.4, 2022.1	Python, C++ languages with capabilities supported in data_dev

NOTE Try out an **Builder with Base Image** example for cpp application source:

- GIT Repo Url: <https://github.com/intel/DevCloudContent-git>
- Resource Name: `openvino-cpp`
- Sub-directory: `Cpp-OpenVINO-sample/`

Select (Builder with Base image) Catalog Name `OpenVINO` and Version `2021.4` from drop-down lists with the `CPP` option.

- Pre Build Script: `pre-build-script.sh`
- Working Dir: `open_model_zoo-2021.4/demos`
- CMAKE Args: `-DCMAKE_BUILD_TYPE=Release`
- CMAKE Build Args: `--target crossroad_camera_demo`
- Post Build Script: `post-build-script.sh`
- Entry Point: `run.sh`

Build process will take a few minutes. Once ready, assign the resource to a new project `openvino-cpp-project` using the Actions in your resources view. Before launching, use [Configure imported containers](#) to view the results in your filesystem.

- Output Mount Point: `/data/`

Above `openvino` dependency `cpp` container builds an executable using `cmake`, downloads and converts the required models. Upon launching, the container performs inference in runtime and saves the resulting labeled video in the filesystem to **download and view**.

You can also try out an **Builder with Base Image** example for python application source:

- GIT Repo Url: <https://github.com/intel/DevCloudContent-git>
- Resource Name: `openvino-python`
- Sub-directory: `Python-OpenVINO-sample/`
- Builder Image with Dependency: `Python`
- Pre Build Script : `pre-build-python.sh`
- Entry Point : `run_python.sh`

Similar to `cpp` example, make sure to configure **Output Mount Point** as `/data/` before launching to retrieve results. The `python` example demonstrates face detection `mtcnn` from the `open-model-zoo` repository. The models are downloaded and converted as pre-build steps and only inference is executed during container runtime.

View build status

Select the **My Library** option to continue monitoring the build status in the background from your resources view.

Dashboard My Library Marketplace Coding Environment Documentation

Import From Dockerfile

Image Name	Source	Status
ovtf:latest	https://github.com/DevcloudContent/dockerfile-import	● Building ⓘ

My Library >

From your **Resources** view, use the **Actions** option next to your *Resource Name* to view build logs, change build configuration or rebuild. Your status will can range from **Building**, **Ready** to **Error**.

DevCloud > Launch > Container Playground

Dashboard My Library Marketplace Coding Environment Documentation

Search by Name

Projects Resources Filesystem

Resource Name	Last Updated	Type	Size	Status	Project
ovtf:latest	Oct 24, 2021	Source	NA	● Building	Unassigned

Once your resource status shows **Ready**, you must **Assign** the container to an existing project to additionally configure container runtime options and launch.

See [Configure imported containers](#) and [Select Hardware and Launch Containers](#) for more info.

Develop Containers in JupyterLab

Use this section for any interactive development tasks before testing your containerized applications on Intel Platforms. You can run jupyter notebooks to get familiar with the OpenVINO™ Inference Engine API and access tools such as model optimizer or open model zoo downloader.

If you plan on writing dockerfiles or any application logic this section also covers how to build containers from the terminal.

- [Run OpenVINO™ notebooks](#)
 - [Navigate to CLI](#)

- [Open the OpenVINO™ notebooks Menu](#)
- [Configure and run notebooks](#)
- [Access OpenVINO™ utilities](#)
 - [Navigate to CLI](#)
 - [Activate virtual environment](#)
 - [Access OpenVINO™ Tools](#)
- [Build containers from terminal](#)
 - [Navigate to CLI](#)
 - [Build a Container](#)
 - [Push to Private Registry](#)
 - [Customize a Sample Application](#)

Run OpenVINO™ notebooks

Your JupyterLab workspace comes installed with OpenVINO™ dependencies accessible in notebooks. Use this development environment to ensure code containing any openvino api runs. For any performance evaluations please see [Test Containers](#).

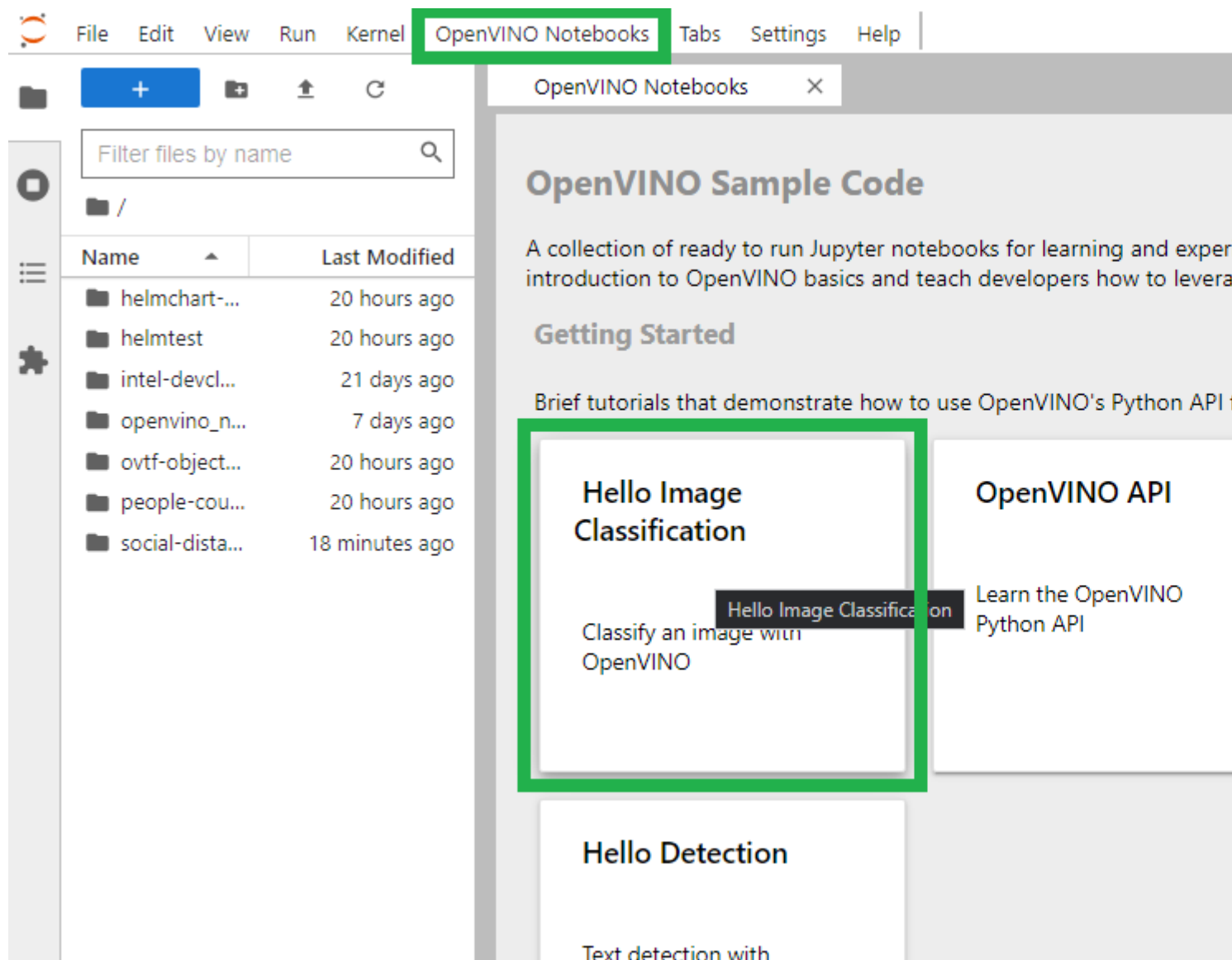
Navigate to CLI

Use the top navigation menu to access the **Coding Environment** to open the JupyterLab interface in a new browser tab. Use the + button from the jupyterlab file browser to open a **Terminal** from the Launcher.

If you not able to access the JupyterLab interface, make sure to **Allow Pop-ups** in your browser.

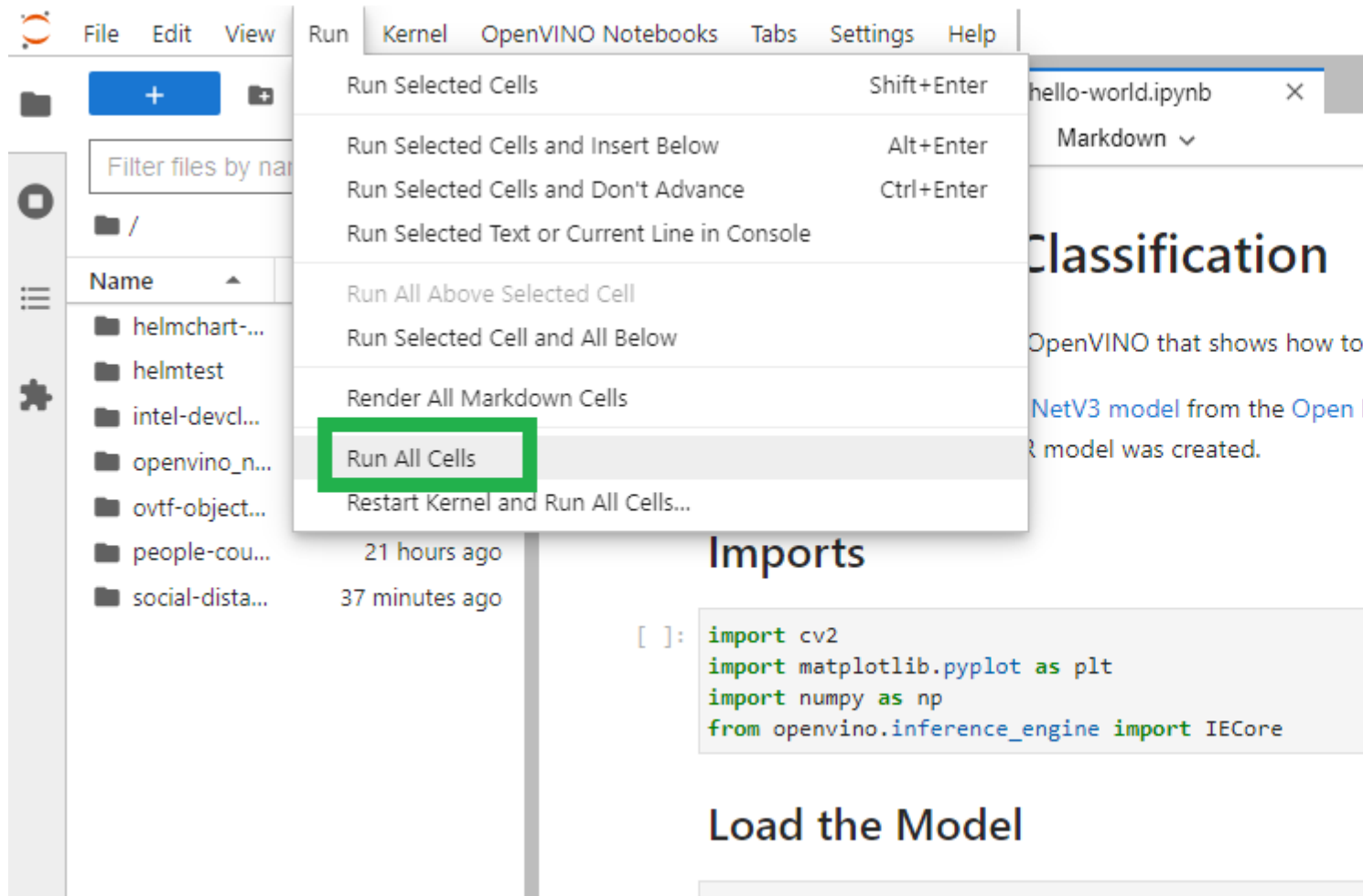
Open the OpenVINO™ notebooks Menu

Use the top menu to see the list of notebooks supported and select a sample to open a notebook in a new tab inside the JupyterLab interface.

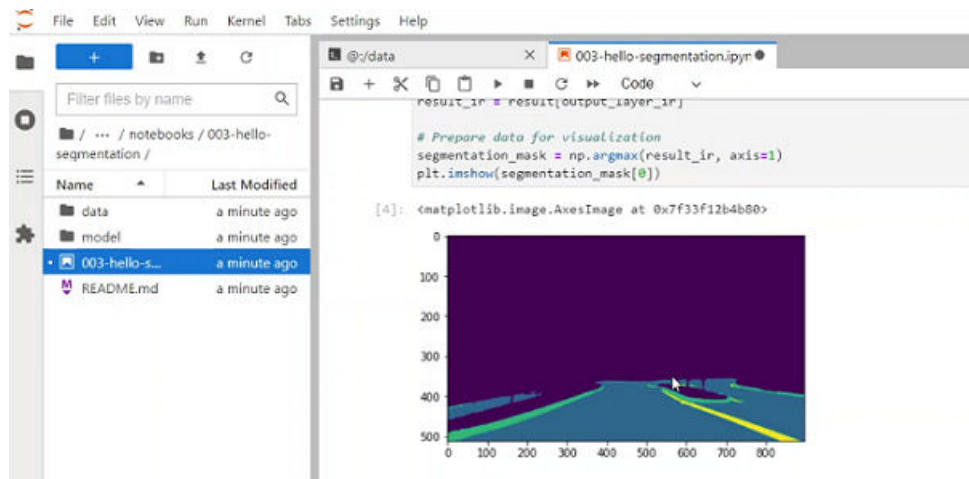


Configure and run notebooks

Make sure a valid OpenVINO kernel is selected on the top right corner of the Jupyter* Notebook and run each cell individually with **SHIFT+ENTER** or use Run > Run All Cells option from the menu in the top-left hand side of your jupyterlab* instance.



Using the above mechanism you can try out other notebooks from the **openvino_notebooks** repository to get familiar with OpenVINO™ api.



NOTE

- Your active Jupyterlab instance can be used for up to **2 hours** before a relaunch is required by clicking **Coding Environment**.
 - Only the Notebooks listed on the menu are officially supported.
 - Additionally, OpenVINO™ developer utilities can be accessed using `!omz_downloader --print_all`, see the [openvino-dev python](#) package for more info.
-

Access OpenVINO™ utilities

One of the common use-cases in the container development workflow is to onboard models trained in popular frameworks such as Caffe, Tensorflow, MXNet or ONNX before building or running your containers. Your development environment comes installed with OpenVINO™ tools such as Model Optimizer, Accuracy Checker, Post-Training Optimization Tool (POT) or downloader and convertor utilities from the OpenVINO™ Model Zoo.

Above tools can be accessed using the installed Python virtual environment from the terminal.

Navigate to CLI

Use the top navigation menu to access the **Coding Environment** to open the JupyterLab interface in a new browser tab. Use the **+** button from the Jupyterlab file browser to open a **Terminal** from the Launcher.

If you not able to access the JupyterLab interface, make sure to **Allow Pop-ups** in your browser.

Activate virtual environment

In a new JupyterLab terminal, **Activate** the `ov2022.1.0-venv python3` virtual environment.

```
source /opt/ov2022.1.0-venv/bin/activate
```

Your terminal session will reflect the name of your activated virtual environment and the shell will begin with `(ov2022.1.0-venv) [build@cliservice-.....]`.

Access OpenVINO™ Tools

Use the short-hand names listed in the [openvino-dev python](#) package. For example, the model downloader tool can be accessed with below command:

```
omz_downloader --print_all
```

For best practices, always **Deactivate** your virtual environment after use with the `deactivate` command.

NOTE

- Use `pip install <package name> --user` to install python packages in the virtual environment.
 - For more information on the all the capabilities and short-hand names of the tools, refer to the [openvino-dev python](#) package.
-

Build containers from terminal

This section shows how to build containers manually from dockerfiles present in your Intel® Developer Cloud for the Edge filesystem inside the JupyterLab interface. Containers built in your JupyterLab terminal session can then be pushed to your private registry associated with your Intel® Developer Cloud for the Edge Container Playground account to test on various Intel Platforms.

NOTE See [Import Code and Build Containers](#) to import source code already hosted in a git repository with just a few clicks using the web user interface.

Navigate to CLI

Use the top navigation menu to access the **Coding Environment** to open the JupyterLab interface in a new browser tab. Use the **+** button from the JupyterLab file browser to open a **Terminal** from the Launcher.

If you not able to access the JupyterLab interface, make sure to **Allow Pop-ups** in your browser.

Build a Container

Use the JupyterLab terminal to build the container with **buildah**:

```
buildah bud --format docker -f DOCKERFILE_NAME.Dockerfile -t CONTAINER_NAME:CONTAINER_TAG .
```

The list of containers built from the JupyterLab interface can be viewed with the `buildah images` command. You can also run `buildah bud --format docker -t CONTAINER_NAME:CONTAINER_TAG .` inside the directory containing the file `.Dockerfile`.

NOTE You cannot run the built container inside the Jupyterlab interface. See the next section to first push the container to your Intel® Developer Cloud for the Edge private container registry as a resource and [Test Containers](#).

Push to Private Registry

You must retag your container to begin with the private registry URL associated with your Intel® Developer Cloud for the Edge account. Use the environment variable `$REGISTRY_URL` to rename your container.

```
buildah tag CONTAINER_NAME:CONTAINER_TAG $REGISTRY_URL/CONTAINER_NAME:CONTAINER_TAG
```

To run and test the containers built in the JupyterLab interface inside a project in the container playground portal push your retagged container with the below command.

```
buildah push $REGISTRY_URL/CONTAINER_NAME:CONTAINER_TAG
```

See [Configure imported containers](#) and [Select Hardware and Launch Containers](#) to launch.

Customize a Sample Application

Your Intel® Developer Cloud for the Edge storage filesystem includes the latest source code for the sample applications hosted on Intel® Developer Cloud for the Edge Marketplace. Each sample includes detailed instructions on:

- How it Works?
- Supported Runtime Configurations
- Build and Run commands on Intel® Developer Cloud for the Edge Container Playground and your local system

The **View/Edit Code** option from the Marketplace opens up the readme for each sample in the JupyterLab environment. You can also visit the public repository:

- [Overview, License and list of Samples](#)

- [Sample Readme](#) residing in the directory for each sample application

NOTE Try building a simple container with below example:

- Create and navigate inside a working directory: `mkdir cli-example && cd cli-example`
 - Write a dockerfile for a lightweight container that prints numbers: `(echo FROM python:slim && echo 'ENTRYPOINT ["seq", "30"]') > Dockerfile`
 - Build the container while including private registry url: `buildah bud --format docker -t $REGISTRY_URL/cli-example:latest`
 - Push the container to assign a project and launch: `buildah push $REGISTRY_URL/cli-example:latest`
-

Enable Cloud Storage

The cloud connector feature brings in file sharing capability between Container Playground and Cloud Storage, such as Amazon Web Services* S3 service. You can download files present in cloud storage and use it in the container workload or you can save file from the Container Playground filesystem to Cloud Storage.

Prerequisites

Before you enable cloud storage connector, you must have the following:

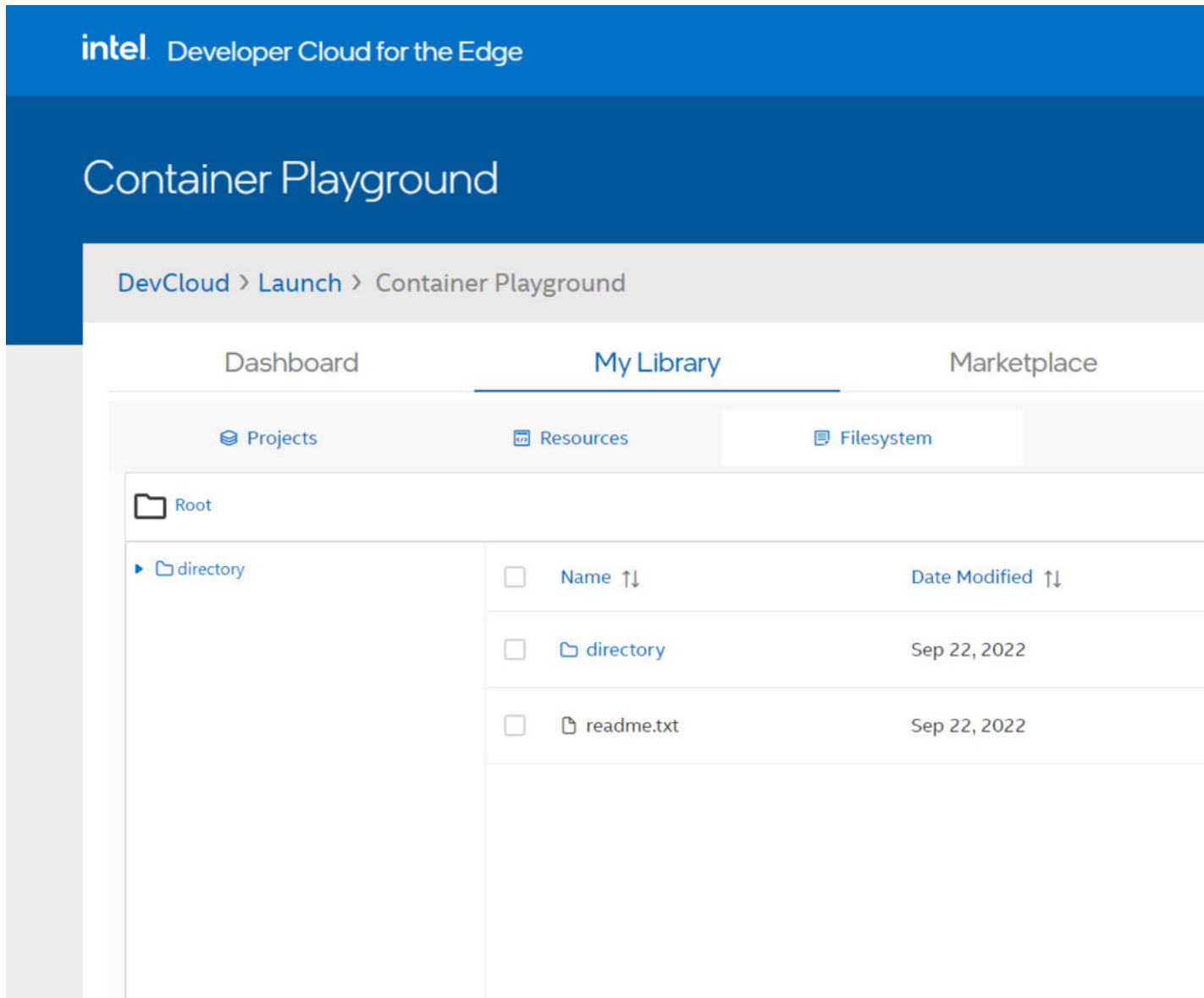
- Amazon Web Services account
- Amazon S3 bucket

For more information, see the following AWS guides:

- Define a user, set up access, and create Secret Key: [Understanding and getting your AWS credentials](#)
- Learn about buckets and regions: [Creating a bucket](#)
- Add files to a bucket: [Uploading Objects](#)

Set Up Cloud Storage Connector


1. From the **My Library** view of the Container Playground, click on the **Filesystem** tab. Click the **Connect** button next to the **AWS S3 Bucket** option.



2. In the **Enter credentials** dialog, enter your AWS Access Key, your AWS Secret Key, select the correct Region, and click **Connect**.

intel. Developer Cloud for the Edge

Enter credentials **AWS S3 Bucket**

 Enter your cloud service provider's credentials to import or export your data

AWS Access Key*

Enter Access Key

AWS Secret Key*


Enter Secret Key

Once connected, you can select one of the buckets listed in the region and click **Submit**.

intel. Developer Cloud for the Edge

Container Playground

Enter credentials [AWS S3 Bucket](#)

 Enter your cloud service provider's credentials to import or export yo

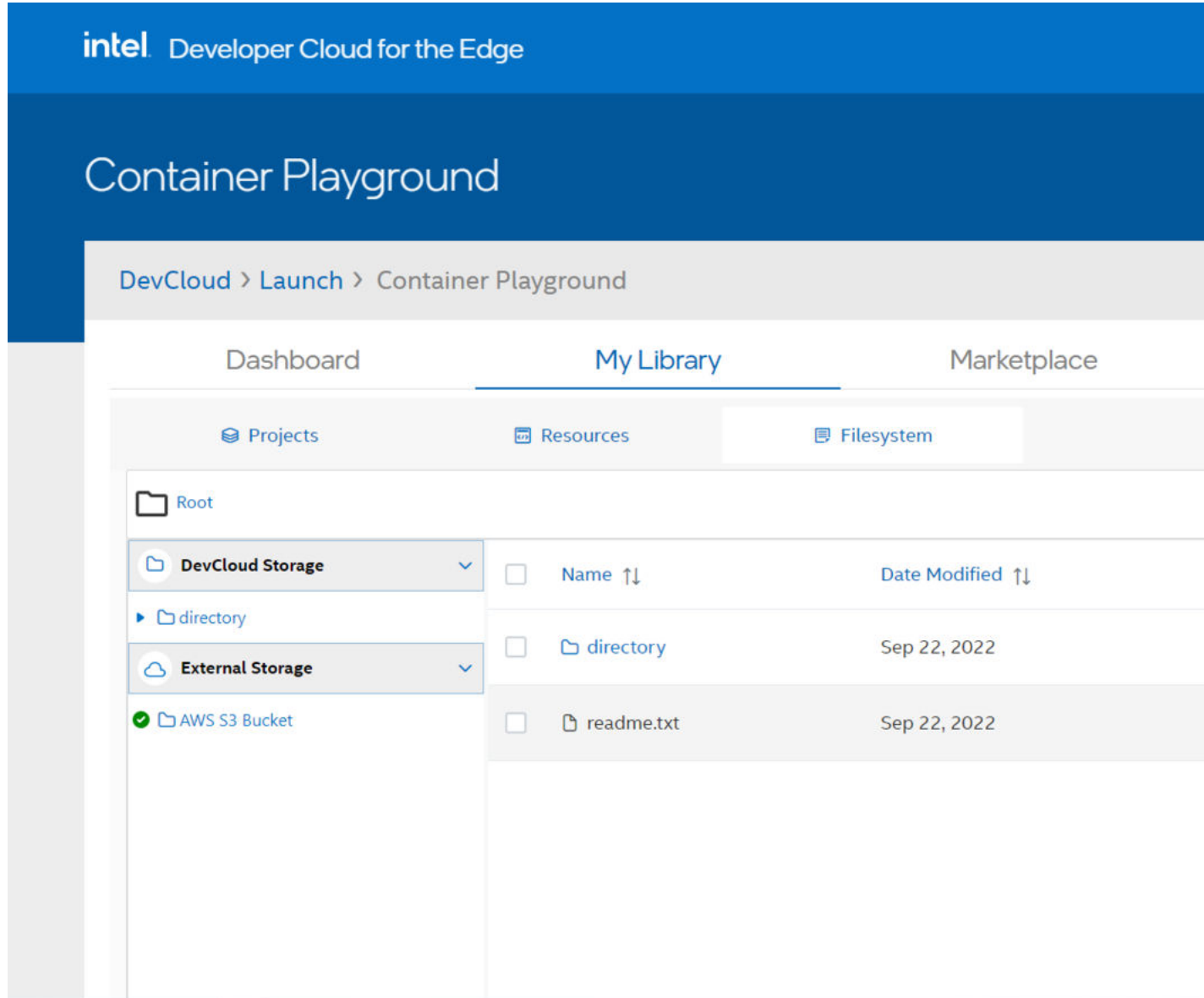
Select the AWS Bucket*

----- Select the bucket -----

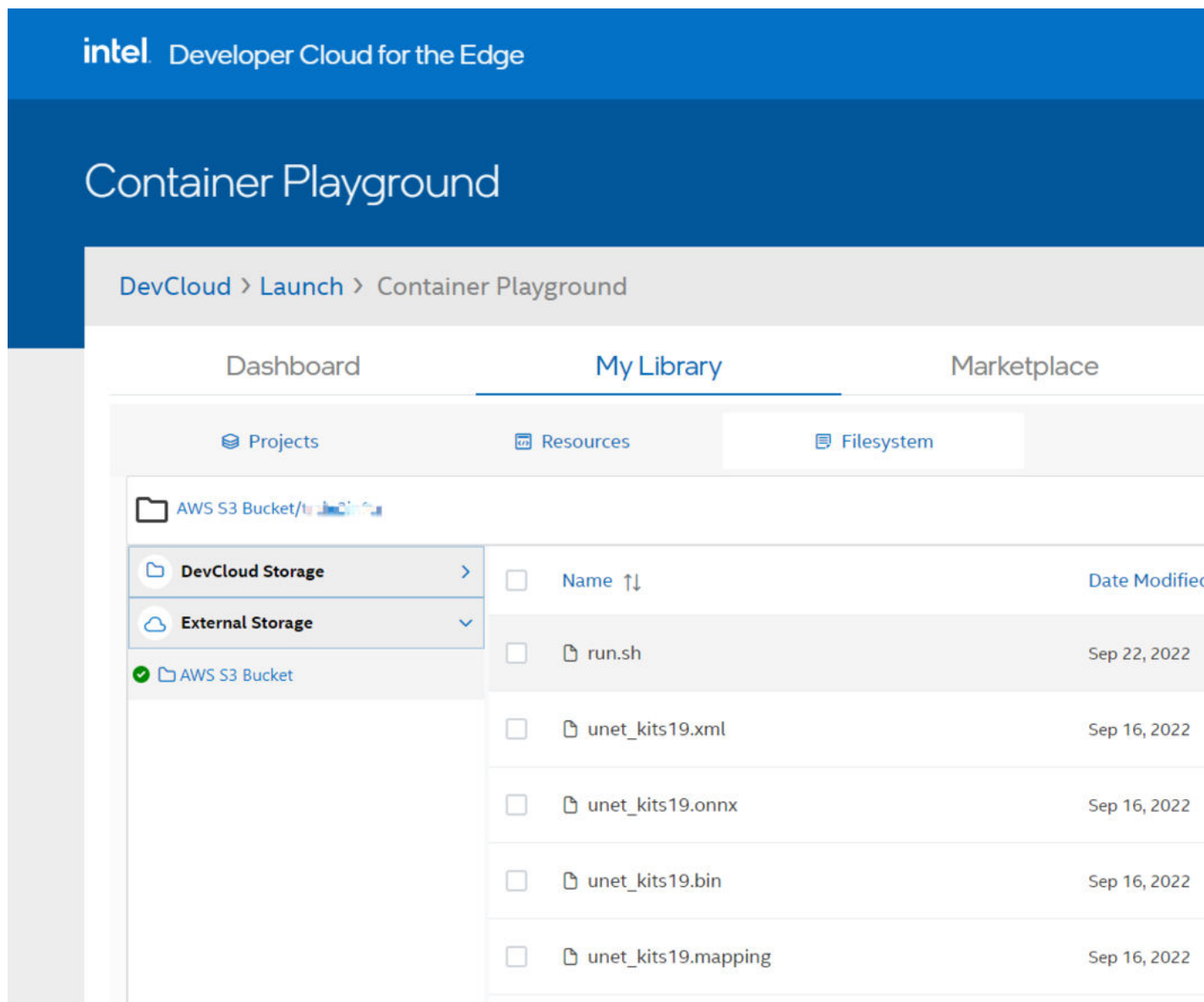
The cloud connector will establish a connection to the AWS S3 Bucket. This may take 2-3 minutes, depending on your network connection.

3. After the connection is completed, you can transfer files between the AWS S3 Bucket and your Container Playground filesystem using the cloud icon.

The image below shows an example of exporting a file from Container Playground to the cloud storage.



The image below shows an example of importing a file from cloud storage to Container Playground.



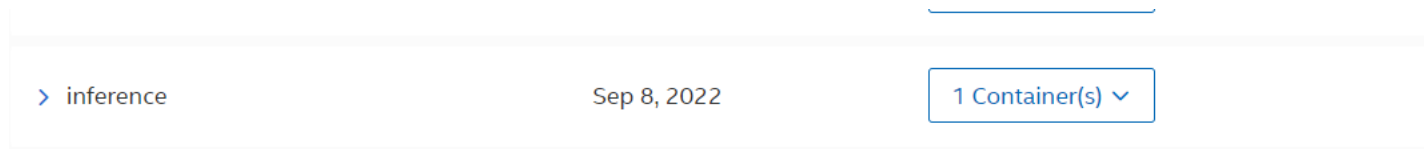
Mount Cloud Storage Files into Existing Container

Refer to [Test Containers](#) for more details on importing and launching containers.

The steps below provide an overview of how an existing container can use cloud storage files.

1. Go to **My Library > Project**.

Click on the **Configure** button for the project where you will mount the imported files.





2. Choose the image.

Click on the **Configure** button for the image where you will mount the imported files.

Configure Project Resources inference

You can select from the list of containers to configure or edit.

Added Containers

Resource Name	Status	Resource Url
 testinference:latest	 Ready	default-route-opensh...
+ Add new Container		

- Mount the path onto image using the following settings:
 - Entry Point:** Use to override the default entryptoint.
 - Volume Binder > Filesystem Path:** Path on the Intel® Developer Cloud for the Edge filesystem (maximum 5). Imported AWS S3 bucket will be saved under `/s3` folder.
 - Volume Binder > Input Mount Point:** Path inside the container (maximum 5).

Configuration **testinference:latest**

Entry Point ⓘ

/s3/run.sh

Output Mount Point ⓘ

/app/results

Volume Binder

Filesystem Path	Input Mount Point
Select Path	Enter container path
/s3/tensorflow	/s3

+ Add new volume binder

Dependency ⓘ

☐ Yes
☒ No

Save the configuration.

4. Access mounted files in the container.

You have configured the image with the mounted volume. When you launch the project, the image will have access to the mounted files/folder.

Available Hardware

Click on the processor number to find out detailed product specifications such as Thermal Design Power (TDP), Processor Frequency, number of Cores and capabilities like Advanced Vector Extensions (AVX) and Vector Neural Network Instruction (VNNI) support.

Intel® Core™ i9 Processors

Processor	Code Name	Product Collection
i9-12900TE	Products formerly Alder Lake	12th Generation Intel® Core™ i9 Processors
i9-12900K	Products formerly Alder Lake	12th Generation Intel® Core™ i9 Processors

Processor	Code Name	Product Collection
i9-11900KB	Products formerly Tiger Lake	11th Generation Intel® Core™ i9 Processors
i9-11900T	Products formerly Rocket Lake	11th Generation Intel® Core™ i9 Processors
i9-11900	Products formerly Rocket Lake	11th Generation Intel® Core™ i9 Processors
i9-10900TE	Products formerly Comet Lake	10th Generation Intel® Core™ i9 Processors

Intel® Core™ i7 Processors

Processor	Code Name	Product Collection
i7-12700	Products formerly Alder Lake	12th Generation Intel® Core™ i7 Processors
i7-11700B	Products formerly Tiger Lake	11th Generation Intel® Core™ i7 Processors
i7-11700T	Products formerly Rocket Lake	11th Generation Intel® Core™ i7 Processors
i7-11700	Products formerly Rocket Lake	11th Generation Intel® Core™ i7 Processors
i7-1185GRE	Products formerly Tiger Lake	11th Generation Intel® Core™ i7 Processors
i7-10700	Products formerly Comet Lake	10th Generation Intel® Core™ i7 Processors
i7-9750H	Products formerly Coffee Lake	9th Generation Intel® Core™ i7 Processors
i7-8665UE	Products formerly Whiskey Lake	8th Generation Intel® Core™ i7 Processors
i7-7700	Products formerly Kaby Lake	7th Generation Intel® Core™ i7 Processors

Intel® Core™ i5 Processors

Processor	Code Name	Product Collection
i5-1145GRE	Products formerly Tiger Lake	11th Generation Intel® Core™ i5 Processors
i5-8365UE	Products formerly Whiskey Lake	8th Generation Intel® Core™ i5 Processors

Intel® Core™ i3 Processors

Processor	Code Name	Product Collection
i3-8145UE	Products formerly Whiskey Lake	8th Generation Intel® Core™ i3 Processors
i3-8109U	Products formerly Coffee Lake	8th Generation Intel® Core™ i3 Processors

Intel® Xeon® Processors

Processor	Code Name	Product Collection
W-1390	Products formerly Rocket Lake	Intel® Xeon® W Processor
W-1290TE	Products formerly Comet Lake	Intel® Xeon® W Processor
Gold-6212U	Products formerly Cascade Lake	2nd Generation Intel® Xeon® Scalable Processors
E-2286M	Products formerly Coffee Lake	Intel® Xeon® E Processor

Frequently Asked Questions

General

- **What's provided when I sign-up for access?**

- Your account will have access for 120 days, with session time of 4 hours per login
- Private registry that can hold up to 15 containers with a total storage limit of 20GB
- Filesystem with a storage limit of 1GB expandable up to 5GB.
- JupyterLab IDE with 2 vCPUs and 4GB RAM with each session lasting for 2 hours
- Exclusive [target platform](#) access for testing containers for up to 30mins
- Up to 8 projects can be created

- **What are some of the security restrictions we need to take care of?**

- Port range must be between 1024 and 65536
- Privileged escalation is not supported
- Host ports are not supported

- **What code editor is provided?**

JupyterLab IDE with 2 vCPUs and 4GB RAM is available for viewing or editing your code, see [Develop Containers in JupyterLab](#).

- **Does Container Playground retain any of my project data after testing containers on a target platform?**

A cleanup procedure is performed after the launched project completes execution or is terminated, before another project can be launched.

- **Can I download the solutions in the Marketplace catalog page?**

Yes, using the [View/Edit Code](#) option for sample applications. You can download some of our reference implementations after signing our licensing terms from Intel Software repo.

Test Containers

- **What are the various public and private registries supported?**

Docker Hub, AWS, Azure and Quay.io public and private registries are supported.

- **What are the supported formats for container filesystem output results?**

The following formats are supported for in browser preview .txt, .mp4, .jpeg, .gif, .png, .csv, and all file formats are available for download.

- **How many projects can be launched simultaneously?**

A project can be launched on up to 3 target platforms simultaneously, and launching of multiple projects is also supported.

- **What is the retention policy?**

The logs are available for 15 days and results in the filesystem can be accessed anytime till the account is active as long as the project is present in your dashboard. The container playground also provides a storage indicator, if you are running low on storage, you can free up space.

- **How long can I run my container applications?**

Upon launching a project, the target hardware is allocated to your user account exclusively until the execution completes or a maximum time limit of 30mins.

- **What is the reason for my launched project to abruptly end?**

Refer to the [View status, output and performance](#) section to learn how to interpret workload status.

- **How can I debug on the Container Playground?**

Container logs, URL routes and terminal debugging is supported, refer to the [View status, output and performance](#) for more info.

- **How can I launch a project that utilizes the Intel® integrated GPUs?**

Projects launched on target platforms that support an integrated GPU, have GPU access enabled by default.

- **Can containers communicate across target platforms?**

Yes, see [Configure imported containers](#) for more info.

- **Why do I see the message "Telemetry not available" on some containers?**

Your project was launched prior to the official performance feature release 2022.2.1, relaunch the project to see performance metrics.

Build Containers

- **How can we build Tensorflow, Pytorch, ONNX, Keras applications?**

Dockerfile would be the best way to build above frameworks, see [Build with dockerfiles](#).

- **How can I re-build sources?**

Any build that is completed can be viewed in `My Library > Resource`. Use the **actions** to edit configuration and provide the latest branch details to start rebuilding.

- **What happens when my github token expires or changes?**

You can add a new token whenever there is a change.

Cloud Connector

- **How can we switch to another AWS S3 bucket from the existing connection?**

Disconnect the current connection by clicking on the **Disconnect** button and follow the steps mentioned in [Enable Cloud Storage](#).

- **Where are the files/folder imported from AWS S3 saved?**

Imported files/folders are saved under the `/s3/<AWS S3 Bucket name>/` folder. You can view them in `My Library > Filesystem`.

- **What files imported from AWS S3 can be saved?**

Any file that is allowed in AWS S3 can be saved.

Known Limitations

General

- Launching projects with the same exposed ports for URL based routes is not supported
- Existing secrets cannot be updated, create a new secret instead
- Preferred screen resolutions are 1920x1080, 1600x900, 1440x900, 1366x768

Test Containers

- Intel® Vision Accelerator Designs with Intel® Movidius™ VPU are not supported in the current release
- For multi-container workloads with inter-container communication, container names are used as service names
- Maximum of 3 different projects can be launched simultaneously.
- Service names must be unique across all resources
- Importing helm-chart can take up to 20 seconds.
- AWS private container image token expires after 12 hours.
- In certain cases, container log output might be out of order or duplicates.
- Spaces are not supported in mount point paths.
- IPV6 ports are not supported.
- "Dashboard Not found" error will be displayed for Project if container is unassigned or deleted.
- In certain cases, GPU name listed on Grafana Performance dashboard does not match Container Playground dashboard.
- CPU consumption data is empty in Grafana Performance dashboard for any workload deployed on Intel® 12th Generation Core™ i9-12900TE platform configuration.

Build Containers

- Building containers from application source without dockerfiles supports C++ only for OpenVINO™ dependency applications
- Gitlab is not supported while importing from source code repository
- Build status will not indicate failure for unsupported languages
- Building containers up to 10GB size are supported
- Current release supports building 1 container at a time
- Maximum of 15 container images can be built while importing from dockerfile or source code hosted on a code repository.

Cloud Connector

- First time connection to AWS S3 fails with Request Failed error. If this occurs, retry the connection again.

About containers, helm-charts and docker-compose

Containers

Applications with minimal dependencies are packaged and distributed in the form of read-only snapshot [container images](#) to ensure your target system environment is always ready to run your application. A running instance of the container image is commonly referred to as a container. This guide and the Container Playground will use **build a container** or **run a container** to simplify and clarify the target use.

Dockerfiles

Containers are built with [dockerfiles](#) that describe all the commands that contribute to the contents (application source, executable, scripts and dependencies) of the container image. For interactive development, the Container Playground coding environment provides **buildah** with usage similar to the [docker commands](#) to build your containers from dockerfiles.

Helm-charts and Docker-Compose

The container playground supports [docker-compose](#), a YAML file describing configurations to deploy one or more container applications. Alternatively, you can deploy with [Helm-charts](#) - a popular mechanism to share, install, and upgrade applications on container orchestration systems like [Kubernetes](#).

NOTE With just a few clicks, you can also use the web user-interface to build containers from your application source code and run your applications on Intel® hardware without any prior knowledge of dockerfiles, docker-compose or helm-charts.

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.