How to create a custom Ocean integration in Port

Ocean is an open-source extensibility framework connecting third-party tools to Port. This is an overview on building a custom integration.

Trevor Garcia Mor Paz

November 20, 2025

← Back

Resources

Blog

Events & Webinars

Resource Center

Day-0 steps before you build an integration

Building your integration

Providing parameters and writing integration code

Running your Ocean integration

Publishing your Ocean integration

Ready to start?

Get a demo

Play with live demo →

Real person here

Listen to article 08:01

Ocean is Port's open-source extensibility framework that makes it easy to connect your existing infrastructure and third-party tools to your Port organization. Integrations are critical to making platform engineering work effectively at scale, and for unifying your software development lifecycle (SDLC) into a single platform.

Ocean is designed to make your Port organization more powerful by giving you ways to build the specific tools you need. In this post, we'll share an overview of how to create a custom Ocean integration, with examples from the Linear integration built by Mor Paz, Director of R&D at Port.

Day-0 steps before you build an integration

There are some important prerequisites to consider before you start building, which will save you time and effort later:

- **Existing integrations:** Do you need to make a custom Ocean integration, or can you use one of Port's out-of-the-box integrations?
- **Admin status:** You'll need to have admin permissions in your organization's portal in order to build an integration. Make sure you have full access before you start.
- **Tool familiarity:** Knowing the tool you're trying to integrate beforehand is a must. Who provides the tool? How does your team use it? What data is exposed by the tool, and which roles should be able to access that data?
- **Day-1 planning:** Defining your team's needs before you start is critical for long-term success. What tools are immediately mandatory for your team? What security and compliance requirements have to be put in place from the outset? What infrastructure is needed, and how should your integration's configuration mapping look?
- **API keys and endpoints:** Gather the specific identifiers you need to connect your portal and the tool you're trying to integrate, and plan your client structure accordingly.

This prep work might seem obvious, but it's important; knowing your integration will work properly and actually serve your organization's needs from the outset will make all the difference when your team starts using it. This is very similar to using a platform-as-a-product approach to IDP development. Without careful preparation, you may find that your integration doesn't do enough for your team, or that necessary use cases have slipped through the cracks.

Building your integration

Ocean comes out of the box with a command-line interface (CLI) and a built-in command to scaffold a new blank integration that you can use as the base for your new integration.

To start, you'll need to have Python 3.11 or higher installed on your machine (tip: use pyenv to manage the Python installations on your machine). From there, follow Port's Ocean documentation to install the Ocean CLI, create your integration project, and set up your developer environment.

```
$ python -m venv .venv

$ source .venv/bin/activate

$ pip install "port-ocean[cli]"

$ ocean new

=====================================================================================
        ::::::::       ::::::::       ::::::::::                :::        ::::
      :+:    :+:     :+:    :+:     :+:                      :+: :+:      :+:+:
     +:+    +:+     +:+            +:+                     +:+   +:+     :+:+:+
    +#+    +:+     +#+            +#++:++#               +#++:++#++:    +#+ +:+
   +#+    +#+     +#+            +#+                    +#+     +#+    +#+  +#+
  #+#    #+#     #+#    #+#     #+#                    #+#     #+#    #+#   #+#
  ########       ########       ##########             ###     ###    ###   ###
=====================================================================================
By: Port.io
🚢 Unloading cargo... Setting up your integration at the dock.
  [1/10] integration_name (Name of the integration): myIntegration
  [2/10] integration_slug (myintegration): my_integration
  [3/10] integration_short_description (A short description of the project):
  [4/10] full_name (Your name): Monkey D. Luffy
  [5/10] email (Your address email <you@example.com>): straw@hat.com
  [6/10] release_date (2023-08-06): 2023-08-06
  [7/10] is_private_integration [y/n] (y): y
  [8/10] port_client_id (you can find it using: https://docs.port.io/build-y
  [9/10] port_client_secret (you can find it using: https://docs.port.io/bui
  [10/10] is_us_region [y/n] (n): y

🌊 Ahoy, Captain! Your project is ready to set sail into the vast ocean of p
Here are your next steps:

⚓ Install necessary packages: Run cd ./my_integration && make install && .
⚓ Copy example env file: Run cp .env.example .env  and update your integrat
⚓ Set sail with Ocean: Run ocean sail to run the project using Ocean.
```

Port resources for your integration

Some config files will need to be added or modified to fit your integration's needs:

- **.port/spec.yaml:** Tells Port what inputs your integration requires, as well as what objects it is able to send to Port.
- **.port/resources/blueprints.json:** Sets up the default blueprints created in Port's data model when your integration is first installed.
- **.port/resources/port-app-config.yaml:** Sets up the default integration mapping used to tell the integration how to parse responses from the third-party tool you're integrating into Port entities. You can learn more about the structure of the port-app-config.yaml file with our mapping configuration docs.

These files will provide structure for your integration and help ensure the finished integration performs as intended. For the Linear integration, the content of the .port/resources/port-app-config.yaml file is as follows:

```
createMissingRelatedEntities: true
deleteDependentEntities: true
resources:
  - kind: team
    selector:
      query: "true"
    port:
      entity:
        mappings:
          identifier: .key
          title: .name
          blueprint: '"linearTeam"'
          properties:
            description: .description
            workspaceName: .organization.name
            url: "\"https://linear.app/\" + .organization.urlKey + \"/team/\"
  - kind: label
    selector:
      query: "true"
    port:
      entity:
        mappings:
          identifier: .id
          title: .name
          blueprint: '"linearLabel"'
          properties:
            isGroup: .isGroup
          relations:
            parentLabel: .parent.id
            childLabels: "[.children.edges[].node.id]"
  - kind: issue
    selector:
      query: "true"
    port:
      entity:
        mappings:
          identifier: .identifier
          title: .title
          blueprint: '"linearIssue"'
          properties:
            url: .url
            status: .state.name
            assignee: .assignee.email
            creator: .creator.email
            priority: .priorityLabel
            created: .createdAt
            updated: .updatedAt
          relations:
            team: .team.key
            labels: .labelIds
            parentIssue: .parent.identifier
```

Providing parameters and writing integration code

Ocean comes out of the box with sane defaults that mean only a minimal set of parameters are required to get the integration up and running. After creating your scaffolding, your new integration will include a main.py file, which is the entrypoint used by the integration when it starts. This file includes comments and placeholders showing where and how your code should be added to the integration. The specific functions you need will dictate the coding required to complete your integration.

Refer to the Ocean documentation on integration configuration for specifics, but you shouldn't need any custom configurations if your integration:

- Has a fixed set of resources
- Doesn't need user-defined parameters
- Doesn't need custom configuration for API calls
- Uses standard endpoints with fixed parameters

As you understand and define which environment variables you'll use, you can easily set up these variables by creating a variables.sh file that includes the different variables you need. This way, when you run the file, it will always make sure the required variables are loaded.

In the case of the Linear integration, Mor used the following in his file:

```
export OCEAN__PORT__CLIENT_ID='YOUR_PORT_CLIENT_ID'
export OCEAN__PORT__CLIENT_SECRET='YOUR_PORT_CLIENT_SECRET'
export OCEAN__EVENT_LISTENER__TYPE='POLLING'
export OCEAN__INTEGRATION__CONFIG__LINEAR_API_KEY='YOUR_LINEAR_API_KEY'
```

Ocean's framework also has built-in support for multiple event listeners, including webhook events, so your integration can react and perform tasks based on events from both Port and the third-party service you're integrating.

Running your Ocean integration

Once you're ready to test your integration, you have two commands to choose from:

- `ocean run` : Uses the Ocean CLI to run the integration
- `make run`: Uses the command defined in the integration's makefile to run the integration

Both commands achieve the same result, which is to run the integration and show you the various logs reported by the integration. In a newly-developed integration, this is the point where you'll notice logical errors, like:

- Creating blueprints fails because the blueprint schema in the blueprints.json file is wrong
- You're unable to query your third-party tool due to bad requests
- Data doesn't appear in your Port organization, or only some of the entity properties populate, due to issues in the port-app-config.yaml base mapping

You might also see an error about missing parameters required by the integration. If you do, make sure you've exported the environment variables required for your integration (as we covered in the Linear integration example with the variables.sh file).

Publishing your Ocean integration

Once you've got your integration running correctly, there are a few final things to take care of before it can be published. Follow these steps to finish the process of creating your integration:

- **Code cleanup:** Verify there are no unused variables, imports, functions, unnecessary code comment blocks, prints, or debug logs. Ensure that all logs use the correct log level and all functions have a correct signature.
- **Linter check:** Run the custom Ocean project linter using the `make lint` command. It will alert you if there are issues that need to be addressed before opening a pull request to the upstream Ocean repository.
- **Open a pull request:** Once any issues in the linter check have been resolved, make sure your fork is synced with the latest version of Ocean's repository, and your integration is using the latest version of Ocean's library. Then, open your request and the Ocean team will review.

The Ocean team at Port may have suggestions or requests for fixes, if applicable. Once those have been addressed, they'll approve and merge the integration, after which an automatic CI/CD process will be triggered that publishes your Ocean integration. Once published, you'll be able to see your Ocean integration in the data sources menu in Port!

Every team is different. Port lets you design a developer experience that truly fits your org.

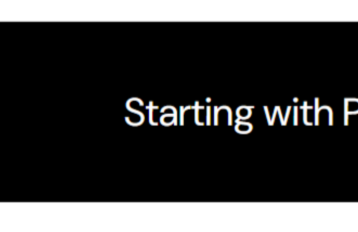
Build your portal

Tags: Guide

← Previous article

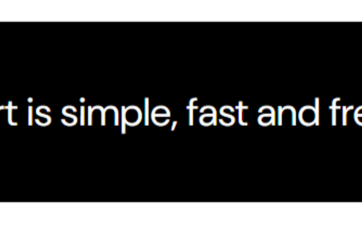
Next article →

You may also be interested in



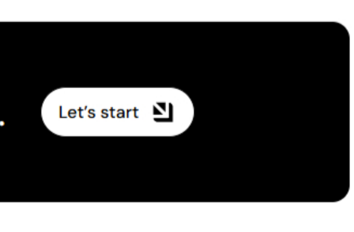
When is the right time to adopt an IDP?

Read more →



Agentic Flow Types: Choosing how you work with AI

Read more →



Why integrations are the core of platform engineer onboarding

Read more →

Starting with Port is simple, fast and free.

Let's start

© 2025 Port.io