Data Science Enabled Business Apps

This document provides an overview of the use case and strategies for integrating Amazon SageMaker Studio with your SaaS application.

It also introduces the Kernel Builder solution, which automates the process of building, managing, and integrating custom environments into Amazon SageMaker Studio. The documentation explains how to use and customize this solution to accelerate your integration efforts. Follow this link if you're only interested in the Kernel Builder solution guide.

Use Case

SaaS providers want to integrate Amazon SageMaker Studio into their applications so that they can provide data scientists with an environment with access to their application data for data science projects. For instance, some marketing teams have access to data scientists. They would like to quickly provide their data scientists with an environment where they can access data from their CRM and CDP systems so that they can deliver predictive models and insights.

Integration Strategy Overview

There are two main strategies that SaaS providers can pursue to integrate Amazon SageMaker Studio into their application.

The first strategy is to enable your customers to run a customized Amazon SageMaker Studio environment that runs in their AWS account. The second strategy involves embedding Amazon SageMaker Studio into your application and providing a managed service.

The main benefit of the first approach is that **it is simple and requires little engineering effort.** However, this approach has limitations:

- **Your customers have to own an AWS account.** This might not be practical for your customers. For instance, your core customers might belong to a line of business that don't own AWS accounts and they are unable to acquire supporting IT resources. Amazon SageMaker Studio is fully managed, so the operational requirements are minimal, but generally, cloud IT support is expected.

- **The user experience is suboptimal.** You should consider scenarios beyond the lack of a unified UI. For instance, if your goal is providing data connectivity out-of-the-box by baking in data drivers into Amazon SageMaker Studio, you should consider network and security snags that the user might encounter. Furthermore, there may also be scale and performance limitations related to how you share the data.

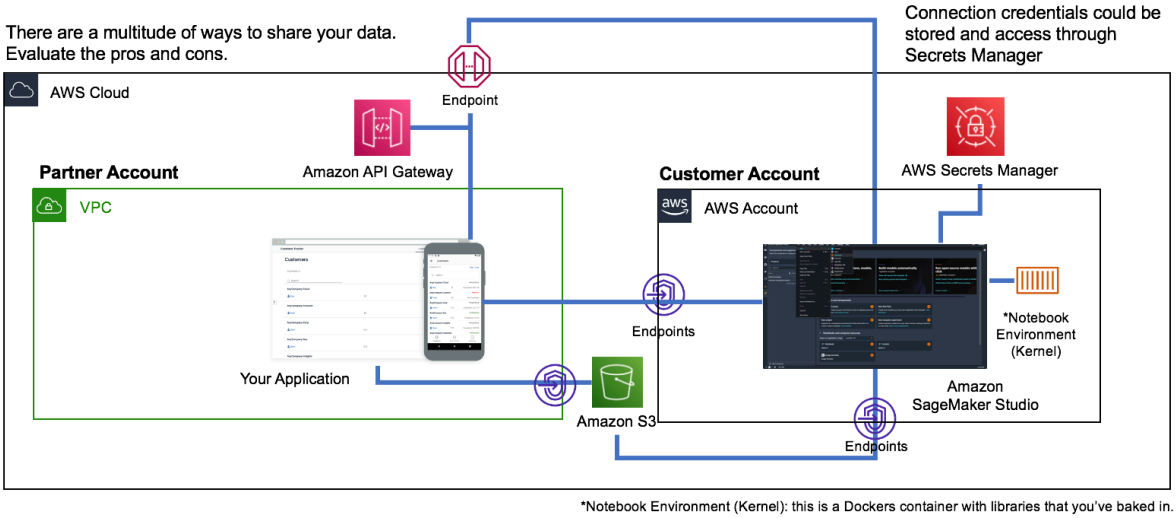
The second approach can address both of the above issues to the degree of your engineering effort and system architecture. If your team isn't ready to invest in the engineering required to deliver a native user experience, you could implement the first strategy to test market demand.

High-level Architectures

1. **Strategy #1:** There are multiple ways to share your application data with your customers through a customer-managed Amazon SageMaker Studio environment. First, provide customers with a custom kernel environment, which has your SDKs and libraries pre-installed.

   For instance, you could bake in a data connector or JDBC drivers into the environment and provide ways to authenticate and connect to your database through a public or VPC endpoint. Alternatively, you could provide mechanisms to unload data into a customer S3 bucket. You would leave your customer with many options like SDKs for Amazon S3, S3 Select, SageMaker DataWrangler, and Amazon Athena to extract and query data within Amazon SageMaker Studio.

   Consider the various trade-offs such as usability, performance, security. Each option has pros and cons and the best option is specific to your circumstances.



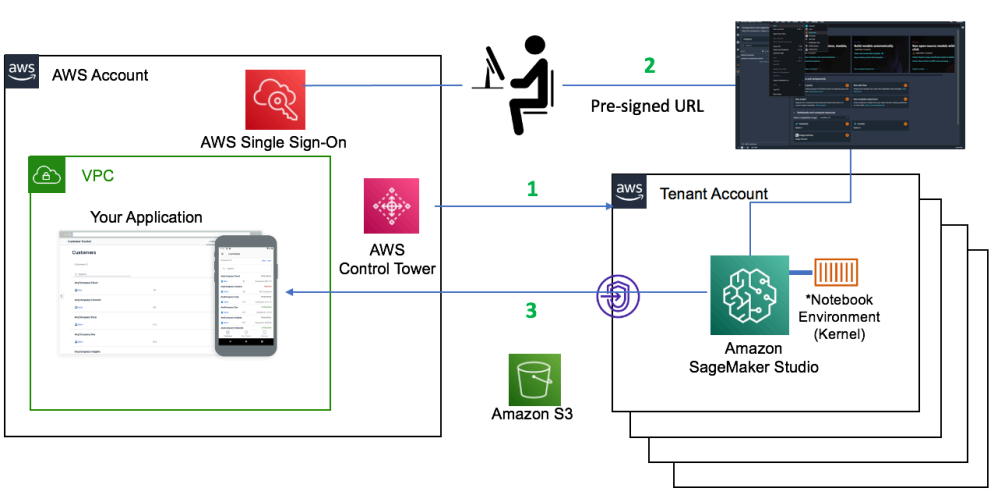
2. **Strategy #2:** Aforementioned, there are benefits to embedding Amazon SageMaker Studio into your SaaS applications, but it will require additional engineering effort.

   First, we advise you to use account-level tenant isolation. Amazon SageMaker Studio provides a collaborative environment for multiple users. However, the most reliable way

to ensure complete resource isolation across your customers is by managing their resources in different AWS accounts. AWS Control Tower is an option for automating the management and provisioning for multiple accounts.

Once the tenant account is onboarded and provisioned, your application could generate a pre-signed URL or use Amazon SageMaker Studio's integration with AWS SSO as the means for providing secure login into the managed environment.

Your custom kernel should be available within the tenant's Amazon SageMaker Studio environment. Your customers can launch notebooks and connect to their data sources using the tools pre-installed in your kernel.



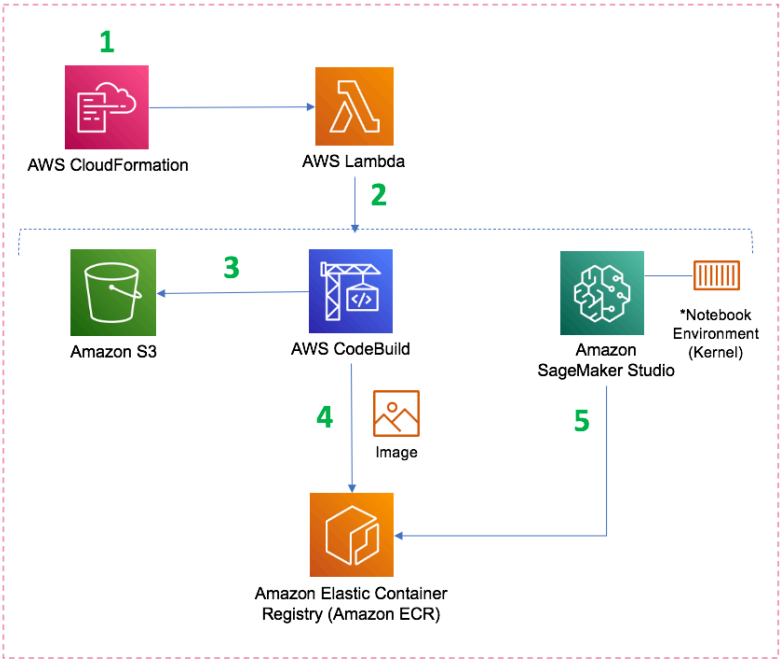
1. Account-level isolation: Control tower provisions an account and Amazon SageMaker environment for each tenant.
2. Users are provided with a pre-signed URL or they log into using SSO credentials.
3. The custom kernel provides out-of-the-box connectivity to the application data.

Integration Kit

Creating a custom kernel involves building, managing, and integrating a Dockers container with Amazon SageMaker Studio. The manual process is documented here.

I have provided an open-sourced solution to automate this process. Under typical circumstances, the integration effort is scoped down to defining a Dockerfile, and solution hardening.

The solution architecture is as follows:



1. CloudFormation provisions the solutions resources.
2. CloudFormation runs Lambda functions that drive the kernel build and publish workflow.
3. CodeBuild references the Dockerfile and buildspec configurations in the workspace to build the custom kernel.
4. The custom kernel is stored and versioned in ECR.
5. The image is registered and attached to the Amazon SageMaker Studio domain.

If you opt for the first integration strategy, this solution provides you with the bulk of the engineering work. This solution has not been hardened. If your engineering team opts to use this solution as part of your production deployment, your team is responsible for ensuring the solution is production-ready.

This solution is also useful for integration strategy 2. Strategy 2 requires additional work to manage authentication and account-level isolation. Nonetheless, this solution can be used to automatically bootstrap each tenant account.

Integrator Requirements

The integrator is required to perform the following steps to refactor the solution for your unique application.

1. Create the Dockerfile for your custom kernel.

   Here's an example of a Dockerfile that creates an environment with Python 3.7, standard Python data science libraries, the SageMaker SDK, and the **Snowflake Python connector**.

```
FROM public.ecr.aws/lambda/python:3.7
RUN pip install sagemaker==2.36.0 && \
   pip install --upgrade snowflake-connector-python[pandas]==2.4.2 && \
   pip install matplotlib==3.4.1 && \
   pip install ipykernel && \
```

```
python -m ipykernel install --sys-prefix && \
pip install jupyter_kernel_gateway
```

2. Copy the Kernel Builder solution contents from s3://dtong-public-fileshare/kernel-builder/ into your S3 bucket. You can run the s3 sync command from the AWS CLI command to do this:

   aws s3 sync *s3://dtong-public-fileshare/kernel-builder/ s3://**<your S3 bucket>**/kernel-builder/*

   Replace **<your S3 bucket>** with the name of an S3 bucket that you own.

3. After the copy process completes, replace the Dockerfile under **s3://<your S3 bucket>/kernel-builder/docker/** with the Dockerfile created in step 1.

4. Your S3 bucket is now ready to serve as the solution repository. Reconfigure the S3 bucket so that your users can access this bucket. Determine who will be deploying your solution and ensure that they have read and list privileges to the contents.

5. The solution is available in three flavors. The "Launch" links below will launch a sample solution that creates a kernel for Snowflake. Create personalized launch button(s) that reference this URL:

   *https://console.aws.amazon.com/cloudformation/home?region=region#/stacks/new?stackName=kernel-builder &templateURL=**<your template location>***

   Deployment options:

   1. **Solution 1:** Full deployment - Launch

      This solution does not require pre-existing resources. It's ideal for new accounts and production environments that use CloudFormation to manage all your AWS resources.

      Key Resources Managed:

      - An S3 bucket to hold assets required by the solution.
      - An Amazon SageMaker Studio environment
      - An ECR repository
      - A build environment for containers (AWS CodeBuild).

      Template location:

*https://**<your S3 bucket>**.s3-**<your bucket's region>.**amazonaws.com/kernel-builder/src/deploy/cf/**kernel-builder-full.yml***

2. **Solution 2:** Build and publish - Launch

   This solution uses a pre-existing Amazon SageMaker Studio environment. Upon launch, it will build your custom kernel and publish it into your Amazon SageMaker Studio environment.

   Key Resources Managed:
   - An S3 bucket to hold assets required by the solution.
   - An ECR repository
   - A build environment for containers (AWS CodeBuild).

   Template location:

   *https://**<your S3 bucket>**.s3-**<your bucket's region>.**amazonaws.com/kernel-builder/src/deploy/cf/**kernel-builder-build-and-publish.yml***

3. **Solution 3:** Publish only - Launch

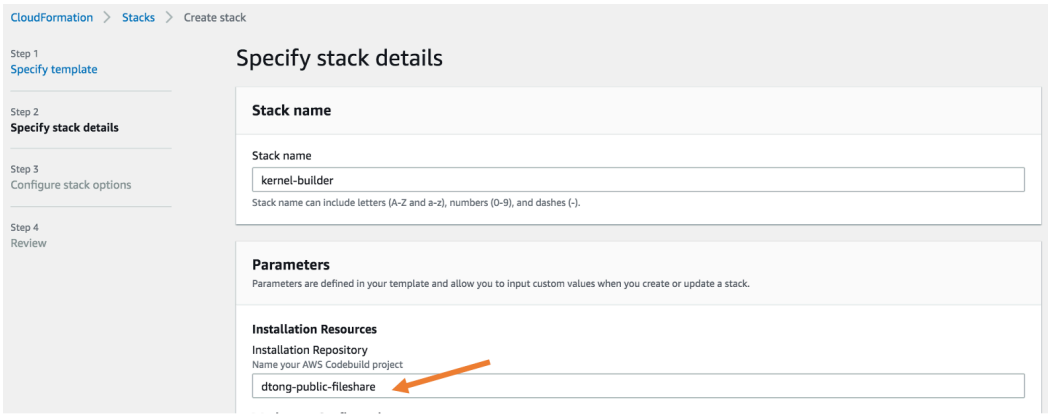
   This solution takes a pre-built kernel image and makes it accessible within a pre-existing Amazon SageMaker Studio environment. This minimal solution is a good option if your custom kernel is already built and available in an ECR repository like the ECR Public Gallery.

   Key Resources Managed:
   - An S3 bucket to hold assets required by the solution.

   Template location:

   *https://**<your S3 bucket>**.s3-**<your bucket's region>.**amazonaws.com/kernel-builder/src/deploy/cf/**kernel-builder-publish-only.yml***

5. The templates use the solution's source bucket as the repository:



Change the default value to your s3 bucket by modifying the following section in each of the templates that you plan to support:

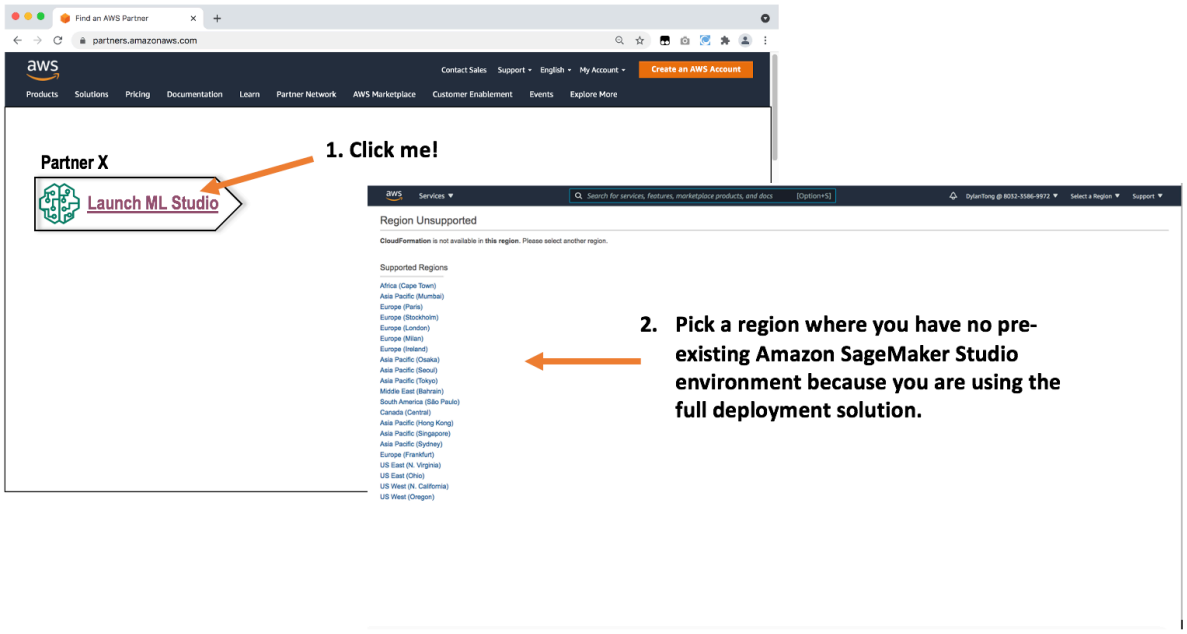
```
pSolutionRepository:
   AllowedPattern: '[A-Za-z0-9-]{1,63}'
   ConstraintDescription: >-
     Maximum of 63 alphanumeric characters. Can include hyphens (-), but not
     spaces. Must be unique within your account in an AWS Region.
   Description:  >-
     Name the S3 bucket hosting the assets required by this solution. The default
     value should be used unless instructed otherwise by the solution provider.
   MaxLength: 63
   MinLength: 1
   Type: String
   Default: dtong-public-fileshare
```

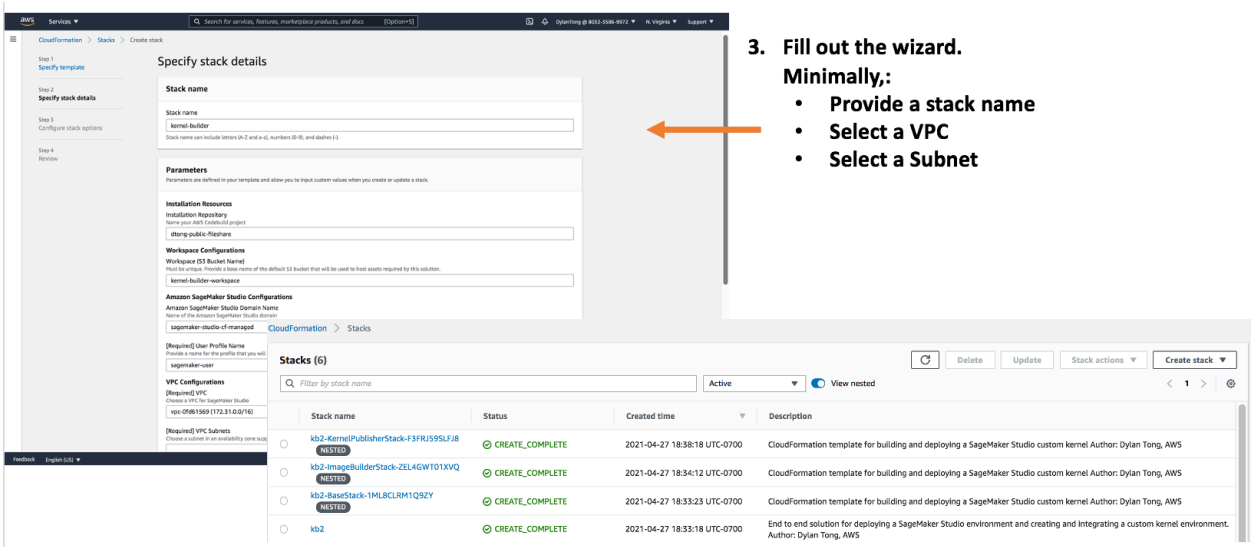
Simply replace the default value “dtong-public-fileshare” with the name of your S3 bucket.

User Flow

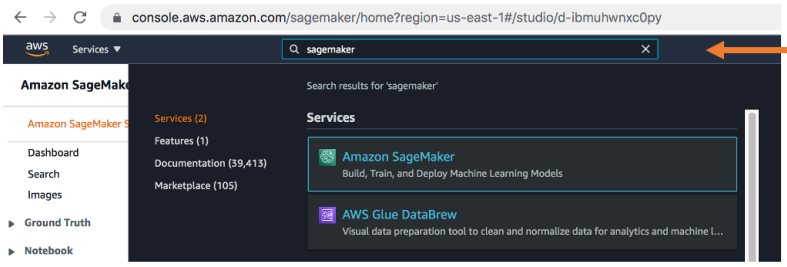
Your solution is ready. Your users can now deploy the solution through the following user experience. As an integrator, you could implement a custom GUI to replace the native CloudFormation wizard.

If you opt to implement strategy 2, the following process could be part of an automated workflow executed through a tenant onboarding system.
Solution Deployment: the following deployment steps could part of an end-user self-service process, or one managed by your customers' cloud IT team.

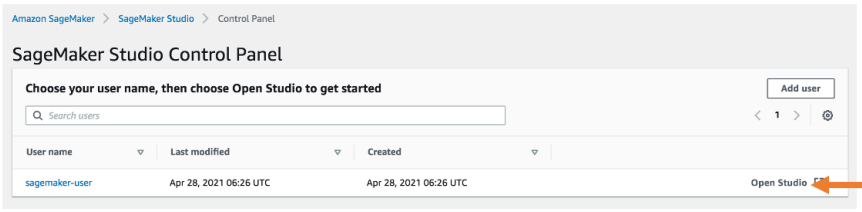




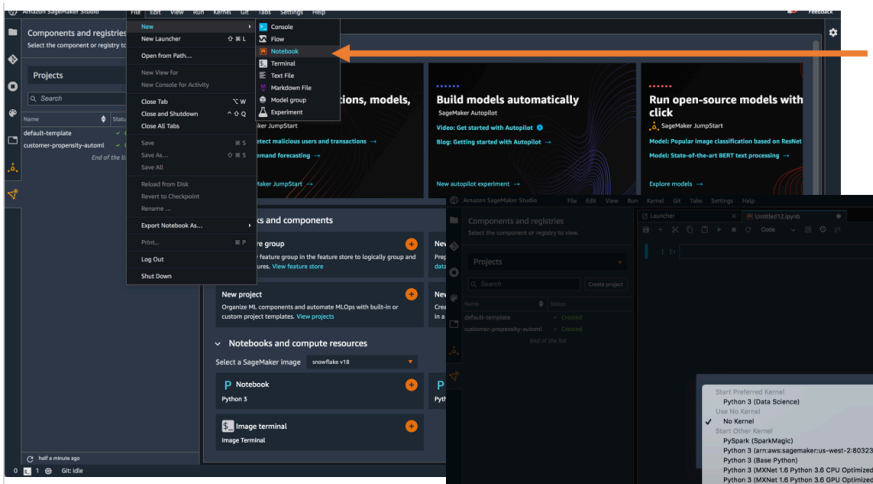
End-user Flow: the following describes the steps that your users will take to access your customized environment.



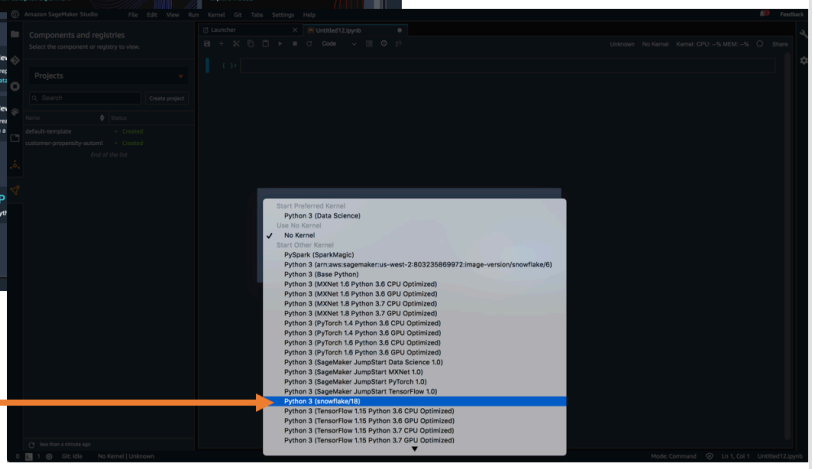
5. **Navigate to the Amazon SageMaker console.**



6. **Launch your Amazon SageMaker Studio environment.**



6. **Launch a new Jupyter Notebook environment.**

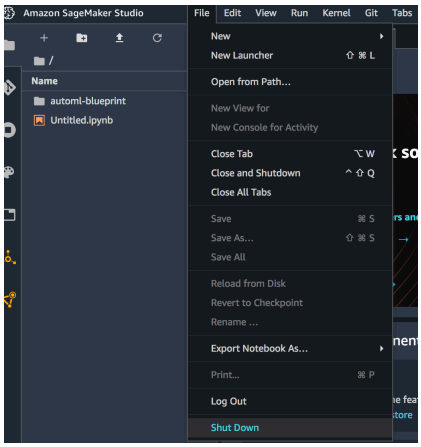


7. **Select your custom kernel. You're ready to explore, experiment and build models in your custom environment!**

FAQ:

1. My custom kernel isn't showing up. What should I do?

Try restarting your Studio environment as a first attempt to resolve the problem.



If you're consistently observing this issue, you can reach out to me: dylatong@amazon.com.

2. **Are there any limitations to the type of custom environments that I create?**

You can learn more about Amazon SageMaker custom kernels here. Environments are built and served as Docker images. The deployment options that automate the image building process are executed and managed by a process hosted on AWS Lambda.

Therefore, your container build process is limited to Lambda's maximum execution time, which is currently 15 minutes. This was a design decision made to provide a solution that is easy to manage and low cost. I will add support for container builds that exceed 15 minutes if there is customer demand. You can make the request by reaching me at dylatong@amazon.com. In the interim, you have the option of building the container and serving it in the ECR Public Gallery, and using the minimal deployment template which doesn't perform a container build at launch time.