Dynamic Discounts in E-commerce

Author(s): Paulo Jerônimo

HTML version
Last updated: 2025-05-25 11:11:26 -0300
Last commit: 08ef1d3

1. Introduction

1.1. The business value of this solution

This solution accelerator allows Snowplow users to learn how their components can be integrated with e-commerce solutions to offer discounts dynamically when detecting an hesitant shopper.

A hesitant shopper is one who has shown a certain amount of interest in a given product without adding it to cart.

This is one of Snowplow's solution accelerators (the second one developed by Paulo Jerônimo @ OSO). Check the first one (snowplow-solution-accel1).

This solution implements two complementary ways of offering discounts:

- The first way to get a discount on a product is to **view it for more than 90 seconds in a 5-minute window**.
  - In the Architecture section, the [ContinuousViewProcessor] implements this feature.
- The second way to get a discount on a product is to **view it more than five times in the same 5-minute window**.
  - In the Architecture section, the [MostViewedProcessor] implements this feature.

After the discount event is generated, one of the situations that can occur in the solution implemented by this accelerator is that it emits the discount event back to the Snowplow pipeline. In the Architecture, this is done by the [DiscountEventSender] component.

In addition to offering this possibility, the processors of this implementation already record the discount event in a Redpanda topic, allowing any consumer with access to this topic to consume it.

When a discount event is available in the Snowplow pipeline, it can be consumed in some other ways, including:

- From the data warehouse, use the Census Dataset API to make the discount available to the front end application
- Using Snowbridge to send the discount to Braze for live couponing.

Detecting high-interest products without purchases is a popular real-time use case for e-commerce.

Although the above definitions seem simple, implementing dynamic discounts are not a trivial task (read more in discounts-processor-doc).

1.2. Technologies used

This project is a companion to the [ecommerce-nextjs-example-store].

It allows you to test it locally, using Development Containers or Docker Compose.

The development of this project can be done entirely in the cloud because it uses Development Containers in GitHub Codespaces. This way, you only need a browser to get started!

This solution was designed so developers can quickly and easily set up their environment to create new features or test existing ones.

Watch the [introduction-video] for details.

What this project does to complement the existence of [ecommerce-nextjs-example-store] is to create the entire infrastructure that allows it to be executed via Docker Compose, or DevContainer + Docker Compose and, ultimately, meet the business requirements.

This solution integrates LocalStack with Snowplow, Redpanda and implements the processors using Kafka Streams.

So, in short, these are the ways to get started:

1. Using GitHub Codespaces ← **the easiest way**.
2. Using your own computer:
   a. Through Development Containers (read Running with DevContainers section) ← **prefer this way**.
   b. Through Docker Compose (read [running-with-docker-compose] section).

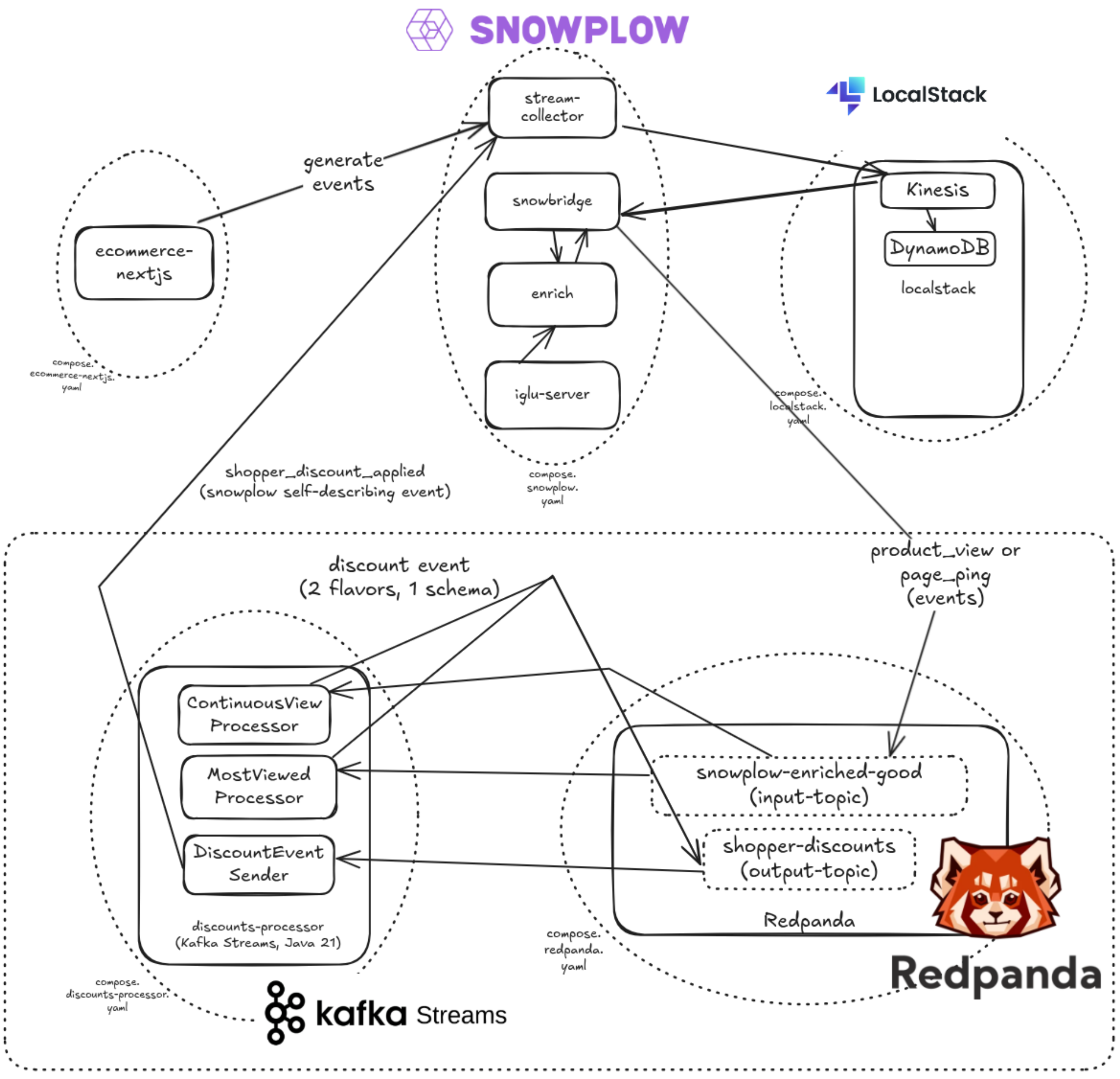
1.3. LICENSE

This is an open-source project.

Check the details in the LICENSE file.

2. Architecture

2.1. Diagram


SNOWPLOW
stream-collector
snowbridge
enrich
iglu-server
compose.snowplow.yaml
generate events
ecommerce-nextjs
compose.ecommerce-nextjs.yaml
LocalStack
Kinesis
DynamoDB
localstack
compose.localstack.yaml
shopper_discount_applied (snowplow self-describing event)
discount event (2 flavors, 1 schema)
product_view or page_ping (events)
ContinuousView Processor
MostViewed Processor
DiscountEvent Sender
discounts-processor (Kafka Streams, Java 21)
compose.discounts-processor.yaml
kafka Streams
snowplow-enriched-good (input-topic)
shopper-discounts (output-topic)
Redpanda
compose.redpanda.yaml
Redpanda

All details are in the next section.

2.2. Components

- The **ecommerce-nextjs-example-store** is a Next.js application that generates tracking events.
    - This application was developed by Snowplow's team before the creation of this solution accelerator.
    - https://github.com/snowplow-industry-solutions/ecommerce-nextjs-example-store.
- The **stream-collector** component sends these events via Kinesis to the **[snowbridge]** component.
- The **snowbridge** component enriches these events, inserts more information (via **[enrich]** component), and sends them to [redpanda].
    - Read more about the **enrich** component here: https://docs.snowplow.io/docs/pipeline-components-and-applications/enrichment-components/enrich-kinesis/.
    - Read more about the **snowbridge** component here: https://docs.snowplow.io/docs/destinations/forwarding-events/snowbridge/.
- The **redpanda** broker receives the events from [snowbridge] in the input-topic (`snowplow-enriched-good`). The events can be from two types: `product_view` or `page_ping`.
- For the following components, read details in the [discounts-processor-doc]:
    - The **ContinuousViewProcessor** component implements the first processor.
    - The **MostViewedProcessor** component implements the second processor.
    - Both processors generates a discount event in the output-topic (`shopper-discounts`).
        - The JSON schema of this event is defined in the file: `schemas/com.snowplow/shopper_discount_applied/jsonschema/1-0-0`.
    - The **DiscountEventSender** component implements the way to send an event back to Snowplow.
        - Basically, it sends the discount event back to Snowplow by sending a POST request to [stream-collector].
        - Please, read the additional [schemas-doc].

All components in this Architecture run as Docker containers via `docker compose`:

- The infrastructure to provide the AWS resources locally (Kinesis, DynamoDB, etc) is created by LocalStack, in the file `compose.localstack.yaml`.
- The Snowplow's components (**[stream-collector]**, **[enrich]**, **[snowbridge]** **and** **[iglu-server]**) are defined in the file `compose.snowplow.yaml`.
- Redpanda's infrastructure is provided by the file `compose.redpanda.yaml`.
- The front-end application (**[ecommerce-nextjs-example-store]**) is defined in the file `compose.ecommerce-nextjs.yaml`.
- The **[ContinuousViewProcessor]**, **[MostViewedProcessor]** and **[DiscountEventSender]** components executed by Kafka Streams which executes a Java application running by Docker through the configuration in the file `compose.discounts-processor.yaml`.

3. Prerequisites for

3.1. Running with DevContainers

Watch the video [introduction-video].

3.1.1. Inside GitHub Codespaces

Watch the video [introduction-video].

3.1.2. Locally, inside VSCode

Watch the video [introduction-video].

3.1.3. Locally, with DevContainer CLI

Step 1 → Install the `devcontainer` command:

```
$ npm install -g @devcontainers/cli
```

Step 2 → Start the dev container:

```
$ # cd TO_THE_PROJECT_FOLDER (the directory where you clone this project)
$ devcontainer up --workspace-folder .
```

Step 3 → Open a Bash terminal in the container:

```
$ devcontainer exec --workspace-folder . bash
```

Please, see more details about how Development Containers is configured in this project by viewing its README file (devcontainer-doc).

3.2. Running with Docker

1. Make sure you have this tools installed:
    a. Bash (version 5.1 or higher). You will run it in a terminal on your macOS, Linux or Windows (in a WSL2 environment).
    b. Git.
    c. Docker (with `docker compose` support).
2. Even though you can run most of the demonstrations for this project using Docker Compose (see docker-doc), in a test environment, in development, or to generate the documentation you are reading for this project, you will need to have a few more tools installed. To run some scripts provided by this project you will need some additional tools like jq, yq, sponge, and docker-asciidoctor-builder.
3. To create a full development environment, you will also need to install:
    a. Java 21.
    b. Node.js 18.
    c. Python 3.12.

> So, here is the best tip to run this project: **it is much more convenient to use Development Containers to setup a development environment because, by creating one, you will have all the required tools installed automatically (inside a docker container)**. So, check Running with DevContainers.

4. Clone this project with Git and cd to it.
5. Execute this command (← **misc-setup**):

```
$ source scripts/misc/setup.sh
```

6. Create a file `docker/.env` (from `docker/.env.sample`) and configure the AWS variables on it.

> *Again…*
>
> **You don't need Java or Node.js configured on your machine if you just want to follow the steps to run the project. You only need a Bash terminal, a Docker installation, and some required tools.**

4. Steps (to run this application as is)

Step 1 → Start the containers

```
$ up.sh
```

Tips:

1. You can press `Ctrl` + `C` at any time. The docker containers will remain running.
2. If there is no file `docker/.env` in the project, this script will try to locate it in a file named `../dynamic-ecommerce-discounts-with-redpanda.env` and copy it to `docker/.env`. This allows you to call `git clean -fdX` at any time you want without losing your configuration.
   a. If the file `../dynamic-ecommerce-discounts-with-redpanda.env` does not exists, it will copy the file `docker/.env.sample` to `docker/.env` and use it.
3. You can pass "services" as an argument option to this script. It will list the options you can pass to it by adding the suffix "-services":

   ```
   $ up.sh services
   apps
   discounts-processor
   ecommerce-nextjs
   localstack
   redpanda
   snowplow
   ```

4. By adding the "-services" to one of the options listed above, you will start only the services listed in the file `compose.<service>.yaml`. So, this will start only the redpanda services (services listed in `compose.redpanda.yaml`):

   ```
   $ up.sh redpanda-services
   ```

5. You can also call the script `up.sh` by using the `compose.sh` script this way:

   ```
   $ compose.sh up
   ```

6. Finally, ***if you can don't want to use this script, you can change your current directory to `docker` and use the `docker` commands you already know***. Check this README file.

Step 2 → Know the URL provided by the services

1. **LocalStack**: https://app.localstack.cloud ← localstack
2. **Redpanda**:
    a. **Internal (docker containers access)** http://localhost:9092 ← redpanda-internal
    b. **Console**: http://localhost:8080 ← redpanda-console
        i. User / password: jane / some-other-secret-password
3. **Ecommerce store**: http://localhost:3000 ← ecommerce-store
    a. It connects with **Snowplow collector** configured to run in http://localhost:9090 ← snowplow-collector

Step 3 → Browse the application pages

As expected, in the [ecommerce-store], during every page navigation, we are tracking a page view event.

For ecommerce interactions we track the following:

- When a customer goes to a product page we track a product view event.
- When a customer sees an internal promotion list, e.g. Homepage promotions, we track an internal promotion view event.
- When a customer clicks an internal promotion, we track an internal promotion click event.
- When a customer goes to a product list page, we track a product list view event.
- When a customer clicks a product on a product list page, we track a product list click event.
- When a customer sees a recommended product list on the product page, we track a product list view event.
- When a customer clicks on a recommended product list on the product page, we track a product list click event.
- When the customer adds a product to the cart, we track an add to cart event.
- When the customer goes to the cart page we track a checkout step event.
- When they go to the payment step, another checkout step event is tracked.
- When the customer successfully completes a transaction, we track a transaction event (triggered on the server-side but formulated with the spec of Snowplow ecommerce)

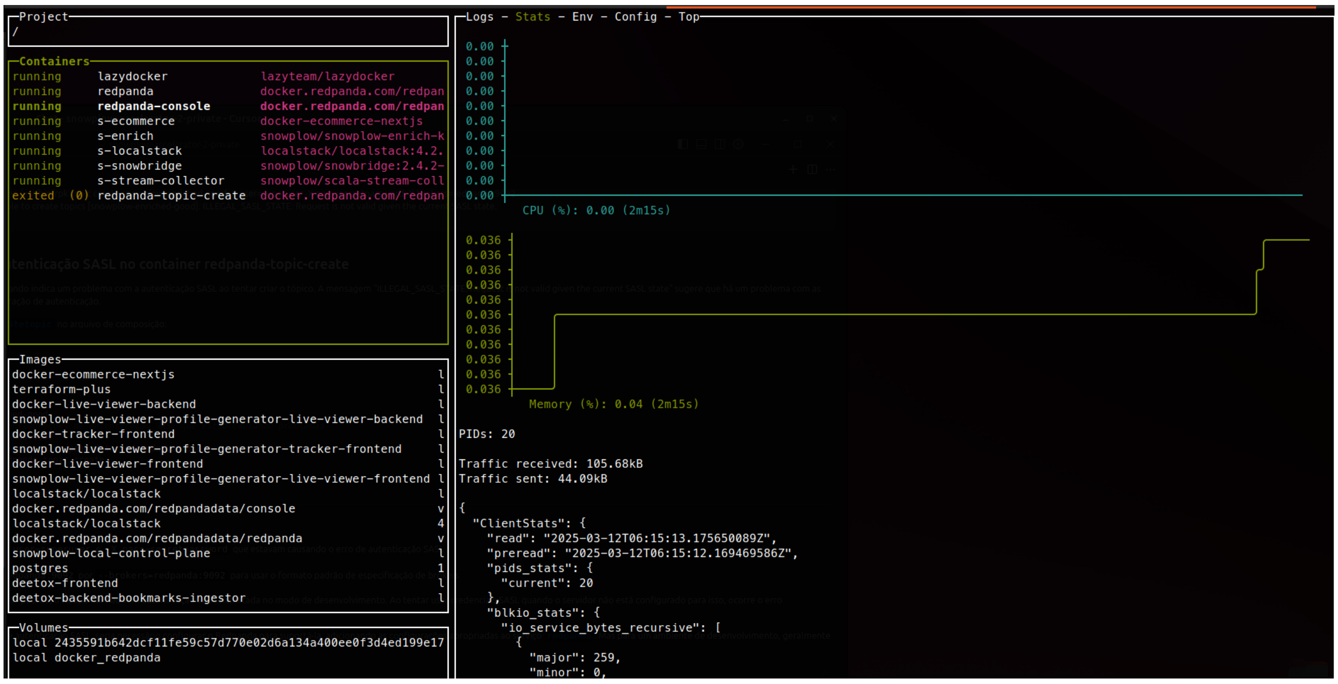
Step 4 → Access the redpanda-console and check the generated events

After browsing the , you can access the and check the generated events in the topic `snowplow-enriched-good`.

Check the [e2e-doc] to see how the steps from 1 to 4 are automated.

Step N → (optional) Use LazyDocker to monitor the containers and logs

```
$ lazy.sh
```



```
Project
/

Containers
running      lazydocker               lazyteam/lazydocker
running      redpanda                 docker.redpanda.com/redpan
running      redpanda-console         docker.redpanda.com/redpan
running      s-ecommerce              docker-ecommerce-nextjs
running      s-enrich                 snowplow/snowplow-enrich-k
running      s-localstack             localstack/localstack:4.2.
running      s-snowbridge             snowplow/snowbridge:2.4.2-
running      s-stream-collector       snowplow/scala-stream-coll
exited  (0)  redpanda-topic-create    docker.redpanda.com/redpan

Images
docker-ecommerce-nextjs                                      l
terraform-plus                                               l
docker-live-viewer-backend                                   l
snowplow-live-viewer-profile-generator-live-viewer-backend   l
docker-tracker-frontend                                      l
snowplow-live-viewer-profile-generator-tracker-frontend      l
docker-live-viewer-frontend                                  l
snowplow-live-viewer-profile-generator-live-viewer-frontend  l
localstack/localstack                                        l
docker.redpanda.com/redpandadata/console                     v
localstack/localstack                                        4
docker.redpanda.com/redpandadata/redpanda                    v
snowplow-local-control-plane                                 l
postgres                                                     1
deetox-frontend                                              l
deetox-backend-bookmarks-ingestor                            l

Volumes
local 2435591b642dcf11fe59c57d770e02d6a134a400ee0f3d4ed199e17
local docker_redpanda
```

```
Logs - Stats - Env - Config - Top

   CPU (%): 0.00 (2m15s)

   Memory (%): 0.04 (2m15s)

PIDs: 20

Traffic received: 105.68kB
Traffic sent: 44.09kB

{
  "ClientStats": {
    "read": "2025-03-12T06:15:13.175650089Z",
    "preread": "2025-03-12T06:15:12.169469586Z",
    "pids_stats": {
      "current": 20
    },
    "blkio_stats": {
      "io_service_bytes_recursive": [
        {
          "major": 259,
          "minor": 0,
```

5. Clean up steps

Step 1 → Stop the containers

To stop all the containers, type:

```
$ ./docker/down.sh
```

Step 2 → Clean up

To remove all the containers and images, type:

```
$ ./docker/clean.sh
```

Warnings:

1. The script `clean.sh` will destroy any data generated by these containers.

6. References

Snowplow

- Snowplow Solution Accelerators
    - Kafka Live Viewer Profiles ← **snowplow-solution-accel1**.

LocalStack

- https://docs.localstack.cloud/
- https://docs.localstack.cloud/user-guide/integrations/devcontainers/ ← **localstack-devcontainers**

Redpanda

- Docker Compose Labs
    - Start a Single Redpanda Broker with Redpanda Console in Docker
- Redpanda Self-Managed Quickstart
- How we engineered our CLI to improve developer productivity
- Some YouTube videos:
    - Why did Redpanda rewrite Apache Kafka? (with Christina Lin)
    - Redpanda Office Hour: HUGE rpk - Redpanda CLI update!

Redpanda Connect

- https://docs.redpanda.com/redpanda-connect/get-started/quickstarts/rpk/
- https://docs.redpanda.com/current/get-started/quick-start/

7. Additional docs

The documentation for this project is designed to be modular, just like all of its code. Therefore, this topic serves as a summary to help you find these additional documents.

1. **devcontainer-doc → .devcontainer/README.adoc**
   DevContainers Configuration
2. **docker-doc → docker/README.adoc**
   Some docker commands used in this project
3. **e2e-doc → scripts/e2e/README.adoc**
   End-to-End (E2E) Testing
4. **discounts-processor-doc → discounts-processor/README.adoc**
   Testing the Discounts Processor Implementation
5. **schemas-doc → schemas/README.adoc**
   Schema implementation and testing

You can always run this command in the root of the project:

```
$ find . -name README.sh
```

It will locate all the scripts that generates additional documents.

8. Demo videos

2025 05 15 16 10 11 → **https://www.youtube.com/watch?v=T_361vfjt9c**

A video demonstrating the execution of e2e tests through docker, in a local environment.

2025 05 12 12 12 15 (introduction-video) → **https://www.youtube.com/watch?v=hA8L2X4ziOk**

A video demonstration the execution of the project (running the e2e tests inside a GitHub codespace). At the time this video was recorded I had an issue that still needs to be resolved in [MostViewedProcessor].

9. Ideas for the next steps

The following ideas describe features or activities that could not be delivered when the demo videos were created. So, they can be considered as tasks to be developed in subsequent releases.

> The order of these ideas was made completely randomly. So, ignore the order below because it must be defined according to Snowplow's needs.

Do some refactoring

One of them concerns the processor code that contains duplicated logic. This can undoubtedly be improved.

Develop a discounts processor version using Redpanda Connect

We tried to implement a solution faster using Bloblang. However, besides forcing a learning curve, testing the solutions with the expected requirements was not easy. Therefore, we abandoned this solution until we had a clearer vision of what we wanted to implement.

Develop a discounts processor version using Apache Flink

We started to develop a solution using Apache Flink. However, we abandoned this solution because we were not able to implement the expected requirements using this technology. Anyway, this is a good task to be developed.

A solution using Apache Flink, running processors in a Docker container in the same way as was created with Kafka Streams, is entirely viable.

Deploy in AWS, via Terraform

The first accelerator developed in partnership with OSO demonstrates this.

Show the discount event directly in the front end

Due to the time spent on the backend and the difficulty in implementing the discount processors, we did not have enough time to complete the development of a visualisation of discount events on the frontend. So this is a good task to be developed.

A quick and straightforward way to implement this is to get the discount event directly from the `shopper-discounts` topic using an API like KafkaJS.

Use TestContainers to implement E2E testing

Since we create some sort of Bash scripts to implement the E2E testing, we decided not to use TestContainers in the first release. However, this is a good task to be developed.

Chat with Paulo Jerônimo (for more ideas)

In addition to the ideas above, Paulo Jerônimo has ideas that may be useful for implementing this solution on different platforms or using other technologies, including AI.