

# DBOS Python

## Build Durable Python Backends That Scale

Qian Li & Peter Kraft, Co-Founders

# Building Reliable Services is Hard

## UNKNOWN\_ERROR

Error: 400620347  
(89F653A8-29D3-4756-  
AEC9-718AA928E859:vHj9Nq8CTlek-  
F8x6JA3Y1w:ios)

OK



We sincerely apologize, but



## Payment Failed

There seems to be a problem with this seller. Please purchase items from an alternative store.  
CSC\_7200065

Try again

Home



r/jetblue · 9 mo. ago  
mrtowser

...

## Seat selection failed but credit card charged



r/jetblue · 2 yr. ago  
Chromaxide

...

## Booked flight through website, haven't received a confirmation email

Question

I booked a flight on JetBlue's website and selected Paypal as a payment method. I paid using Paypal Credit and when I was redirected back to JetBlue's website the page remained blank.

ht, and right after I saw the charge (on), the JetBlue app said seat or record that I have paid JetBlue charge on my Amex. Anyone

# What is DBOS?

1. Make your apps crash-proof with lightweight durable execution
  - If your app is interrupted, it restarts from where it left off
  - Each step executes exactly-once

# What is DBOS?

1. Make your apps crash-proof with lightweight durable execution
  - If your app is interrupted, it restarts from where it left off
  - Each step executes exactly-once
2. Serverlessly deploy your apps to the cloud
  - No resource provisioning – push-button deploy, scale automatically
  - Pay only for the CPU time you actually use

# Get Started!

1. Take your Python app

```
from fastapi import FastAPI

app = FastAPI()

def step_one():
    print("Step one completed!")

def step_two():
    print("Step two completed!")

@app.get("/")
def workflow():
    step_one()
    step_two()
```

# Get Started!

1. Take your Python app
2. Decorate your functions
  - `@DBOS.workflow()`
  - `@DBOS.step()`
  - `@DBOS.transaction()`

```
from fastapi import FastAPI
from dbos import DBOS

app = FastAPI()
DBOS(fastapi=app)

@DBOS.step()
def step_one():
    print("Step one completed!")

@DBOS.step()
def step_two():
    print("Step two completed!")

@app.get("/")
@DBOS.workflow()
def workflow():
    step_one()
    step_two()
```

# Get Started!

1. Take your Python app
2. Decorate your functions

- `@DBOS.workflow()`
- `@DBOS.step()`
- `@DBOS.transaction()`

3. Deploy it to the cloud

```
> dbos-cloud app deploy
```

```
[info]: Successfully deployed dbos-app!  
[info]: Access your application at  
https://demo-dbos-app.cloud.dbos.dev/
```

## Use Cases

We'll show you:

- Payment processing
- AI-powered Slackbot

You can also use DBOS for:

- Scheduled data scraping
- Reliable event processing
- Multi-agent workflows
- Long-running background jobs

## Widget Store Demo

- <https://demo-widget-store.cloud.dbos.dev/>
- <https://docs.dbos.dev/python/examples/widget-store>

## Server Tools

---

Crash the application at any time to simulate a server failure or service interruption. After crashing, the application will momentarily become inaccessible but then resume from exactly where it left off before the crash. Try it and see!

[Crash the Application](#)

```
@app.post("/crash_application")  
def crash_application():  
    os._exit(1)
```

```
@DBOS.workflow()
def checkout_workflow():
    # Create a new order
    order_id = create_order()
```

```
@DBOS.workflow()
def checkout_workflow():
    # Create a new order
    order_id = create_order()

@DBOS.transaction()
def create_order() -> int:
    result = DBOS.sql_session.execute(
        orders.insert().values(order_status=OrderStatus.PENDING.value)
    )
    return result.inserted_primary_key[0]
```

```
@DBOS.workflow()
def checkout_workflow():
    # Create a new order
    order_id = create_order()

    # Attempt to reserve inventory, cancelling the order if no inventory remains.
    inventory_reserved = reserve_inventory()
```

```
@DBOS.workflow()
def checkout_workflow():
    # Create a new order
    order_id = create_order()

    # Attempt to reserve inventory, cancelling the order if no inventory remains.
    inventory_reserved = reserve_inventory()
    if not inventory_reserved:
        logger.error(f"Failed to reserve inventory for order {order_id}")
        update_order_status(order_id=order_id, status=OrderStatus.CANCELLED.value)
        DBOS.set_event(PAYMENT_ID, None)
    return
```

```
@DBOS.workflow()
def checkout_workflow():
    # Create a new order
    order_id = create_order()

    # Attempt to reserve inventory, cancelling the order if no inventory remains.
    inventory_reserved = reserve_inventory()
    if not inventory_reserved:
        logger.error(f"Failed to reserve inventory for order {order_id}")
        update_order_status(order_id=order_id, status=OrderStatus.CANCELLED.value)
        DBOS.set_event(PAYMENT_ID, None)
        return

    # Send a unique payment ID to the checkout endpoint so it
    # can redirect the customer to the payments page.
    DBOS.set_event(PAYMENT_ID, DBOS.workflow_id)
```

```
@DBOS.workflow()
def checkout_workflow():
    ... # Create a new order and reserve inventory

    # Wait for a message that the customer has completed payment.
    payment_status = DBOS.recv(PAYMENT_STATUS)
```

```
@DBOS.workflow()
def checkout_workflow():
    ... # Create a new order and reserve inventory

    # Wait for a message that the customer has completed payment.
    payment_status = DBOS.recv(PAYMENT_STATUS)

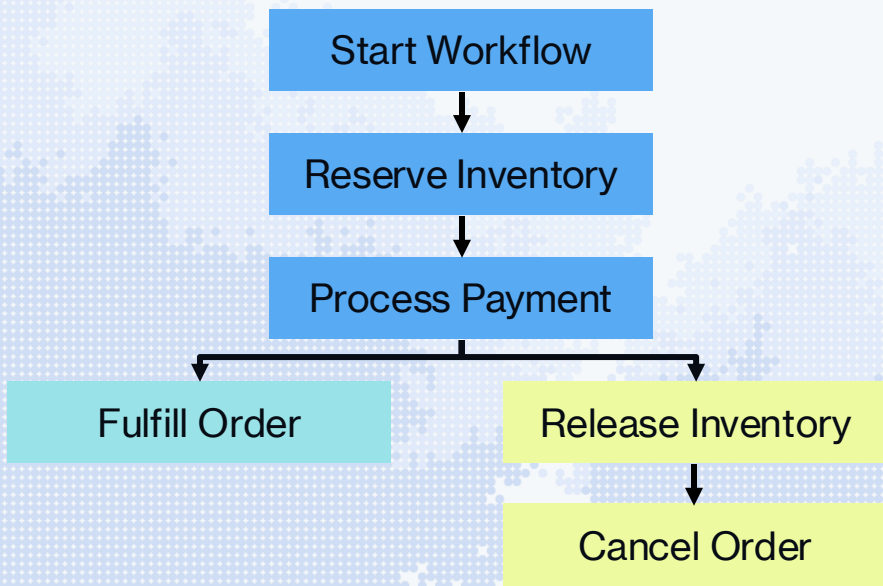
    # If payment succeeded, mark the order as paid.
    # Otherwise, return reserved inventory and cancel the order.
    if payment_status == "paid":
        logger.info(f"Payment successful for order {order_id}")
        update_order_status(order_id=order_id, status=OrderStatus.PAID.value)
```

```
@DBOS.workflow()
def checkout_workflow():
    ... # Create a new order and reserve inventory

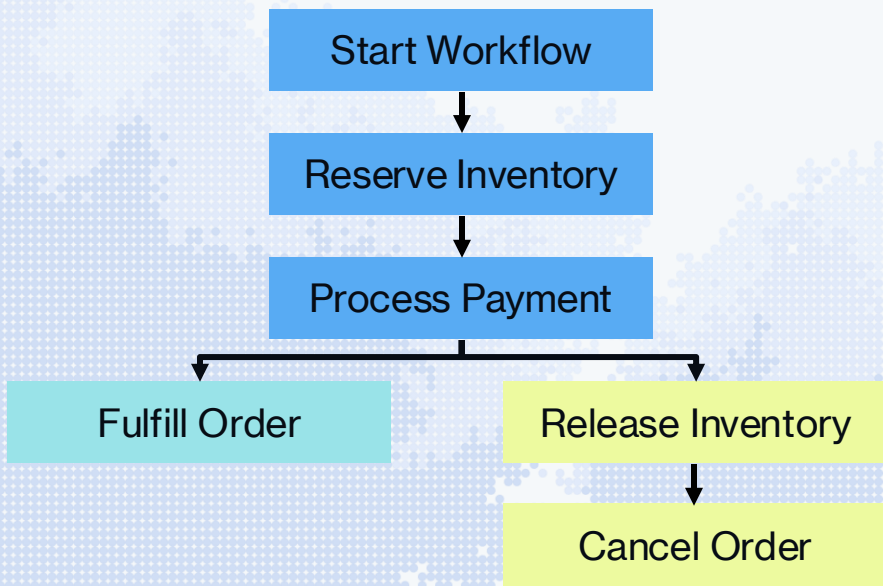
    # Wait for a message that the customer has completed payment.
    payment_status = DBOS.recv(PAYMENT_STATUS)

    # If payment succeeded, mark the order as paid.
    # Otherwise, return reserved inventory and cancel the order.
    if payment_status == "paid":
        logger.info(f"Payment successful for order {order_id}")
        update_order_status(order_id=order_id, status=OrderStatus.PAID.value)
    else:
        logger.warn(f"Payment failed for order {order_id}")
        undo_reserve_inventory()
        update_order_status(order_id=order_id, status=OrderStatus.CANCELLED.value)
```

## How DBOS Workflows Works:



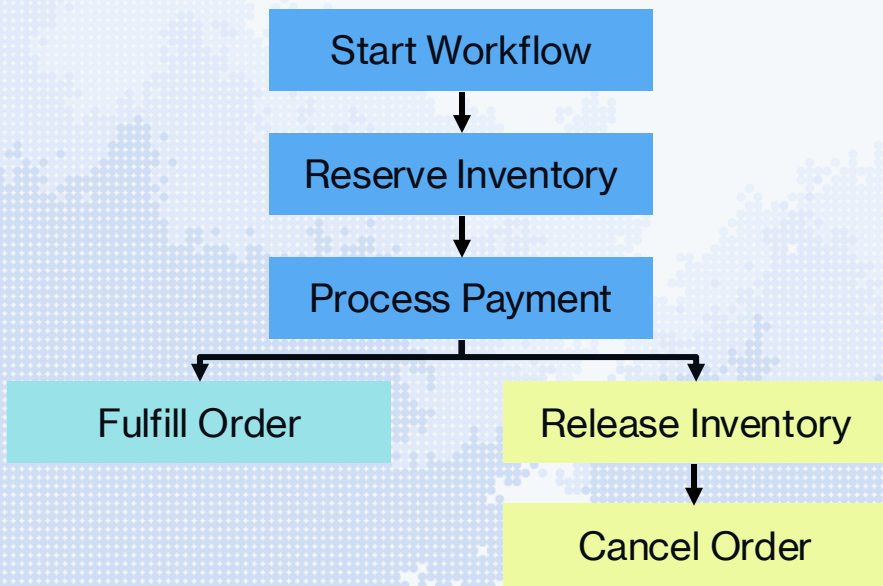
## Record the Workflow Start



ID: hg4ttg2  
Status: PENDING



# Record Each Step's Output

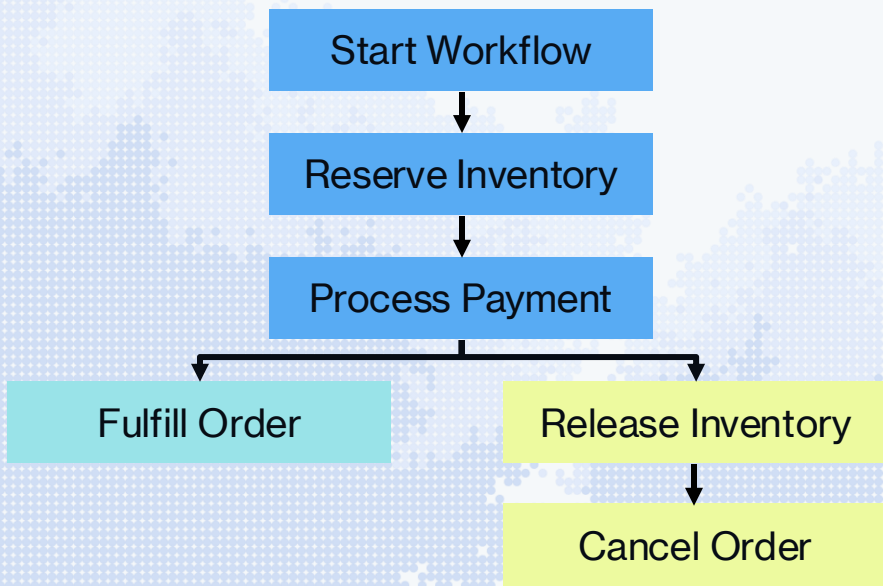


ID: hg4ttg2  
Status: PENDING

Step: 1  
Output: Reserved



## Record Each Step's Output



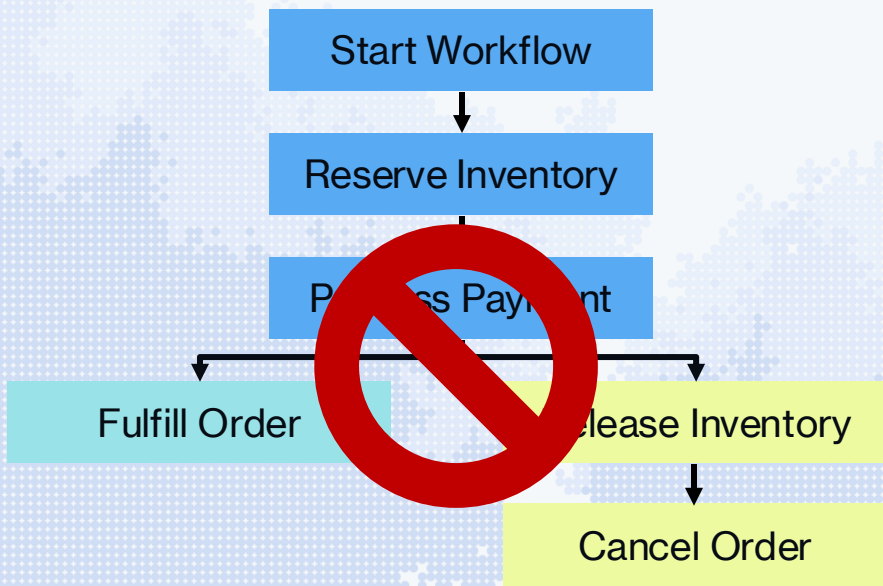
ID: hg4ttg2  
Status: PENDING

Step: 1  
Output: Reserved

Step: 2  
Output: Successful



# What If It Crashes?



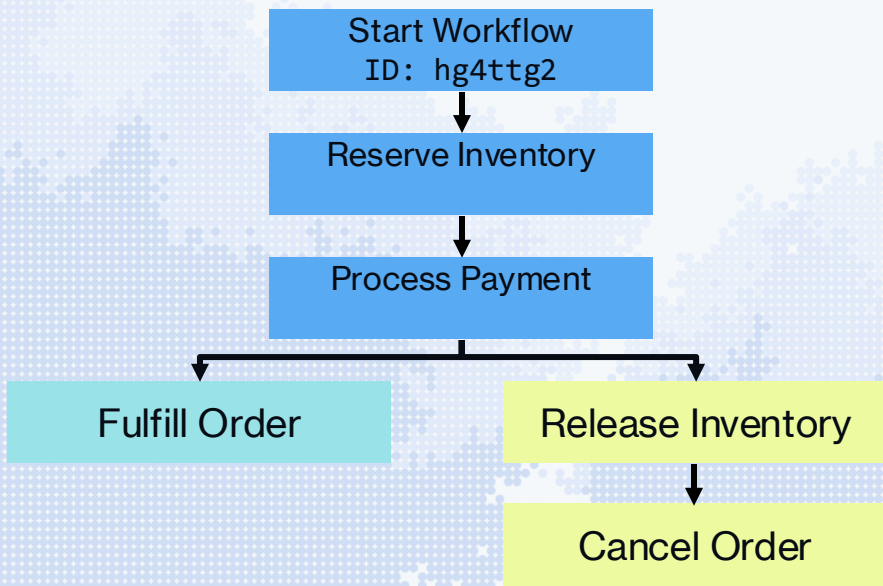
ID: hg4ttg2  
Status: PENDING

Step: 1  
Output: Reserved

Step: 2  
Output: Successful



# Restart and Check Recorded Outputs



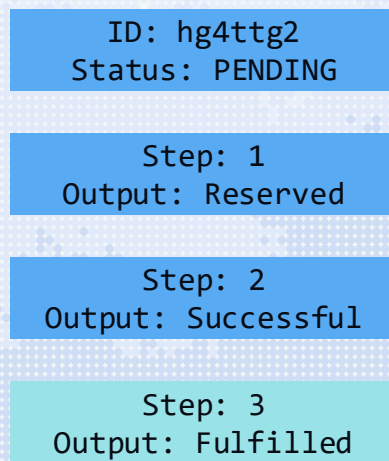
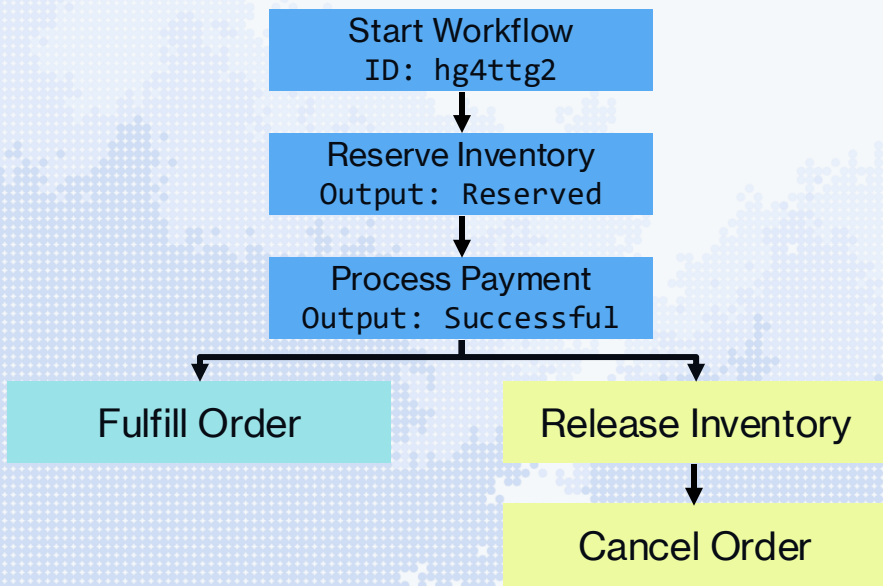
ID: hg4ttg2  
Status: PENDING

Step: 1  
Output: Reserved

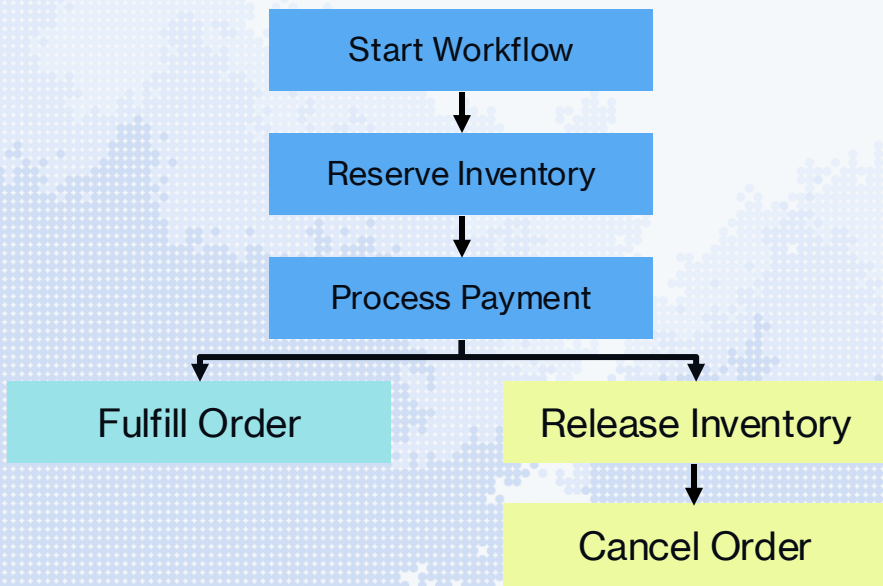
Step: 2  
Output: Successful



# Resume From Where You Left Off



# Resume From Where You Left Off



ID: hg4ttg2  
Status: **SUCCESSFUL**



## AI-Powered Slackbot Demo

- <https://docs.dbos.dev/python/examples/rag-slackbot>



## AI-Powered Slackbot Demo

We started with an off-the-shelf RAG demo from LlamaIndex...

...but no one tells you how to deploy AI apps to production

1. You need to host your app on the cloud
2. You need to handle Slack timeouts and duplicate events
3. At scale, you need to handle rate-limiting

```
@slackapp.message()
def handle_message(request):
    # Use the unique event_id as an idempotency key to
    # guarantee each message is processed exactly-once
    event_id = request.body["event_id"]
    with SetWorkflowID(event_id):
        # Start the event processing workflow in the background then respond to Slack.
        # We can't wait for the workflow to finish because Slack expects the
        # endpoint to reply within 3 seconds.
        DBOS.start_workflow(message_workflow, request.body["event"])
```

```
@DBOS.workflow()
def message_workflow(message):
    # If the message is a question for the bot, respond on the Slack channel
    if is_question(message):
        query = get_text(message)
        response = answer_question(query, message)
        post_slack_message(str(response), message["channel"], message.get("thread_ts"))
```

```
@DBOS.workflow()
def message_workflow(message):
    # If the message is a question for the bot, respond on the Slack channel
    if is_question(message):
        query = get_text(message)
        response = answer_question(query, message)
        post_slack_message(str(response), message["channel"], message.get("thread_ts"))
    # If the message is a reply to the bot, treat it as a question
    elif is_reply(message):
        query = get_text(message)
        replies = get_slack_replies(message["channel"], message["thread_ts"])
        response = answer_question(query, message, replies)
        post_slack_message(str(response), message["channel"], message["thread_ts"])
```

```
@DBOS.workflow()
def message_workflow(message):
    # If the message is a question for the bot, respond on the Slack channel
    if is_question(message):
        query = get_text(message)
        response = answer_question(query, message)
        post_slack_message(str(response), message["channel"], message.get("thread_ts"))
    # If the message is a reply to the bot, treat it as a question
    elif is_reply(message):
        query = get_text(message)
        replies = get_slack_replies(message["channel"], message["thread_ts"])
        response = answer_question(query, message, replies)
        post_slack_message(str(response), message["channel"], message["thread_ts"])
    # If the message isn't a question, store it in the index as a "fact"
    else:
        user_name = get_user_name(message["user"])
        insert_node(message["text"], user_name, message["ts"])
```

```
@DBOS.step()
def answer_question(query, message, replies = None):
    who_is_asking = get_user_name(message["user"])
    template = generate_prompt(message, who_is_asking, replies)
    qa_template = PromptTemplate(template)
    postprocessor = FixedRecencyPostprocessor(
        top_k=20,
        date_key="when",
    )
    query_engine = index.as_query_engine(
        similarity_top_k=20, node_postprocessors=[postprocessor]
    )
    query_engine.update_prompts({"response_synthesizer:text_qa_template": qa_template})
    return query_engine.query(query)
```

## DBOS Features

We've shown you:

- Lightweight durable execution
- Exactly-once event processing
- Serverless deployment

DBOS also provides:

- Automatic workflow tracing
- Scheduled jobs
- Time travel

Coming soon:

- Job queues
- Asyncio support
- Database trigger events

# Learn More

[Home](#) > [Overview](#)

## Featured Examples



### Widget Store



Use DBOS durable workflows to build an online storefront that's resilient to any failure.



### Earthquake Tracker



Use DBOS to build a real-time earthquake dashboard by streaming data from the USGS into Postgres, then visualizing it with Streamlit.



### Hacker News Bot



Use DBOS to deploy a scheduled job that regularly searches Hacker News for comments about serverless computing and posts them to Slack.



### AI-Powered Slackbot



Use DBOS and LlamaIndex to build an AI-powered Slackbot that uses RAG to answer questions about previous Slack conversations.

## Start Building with DBOS

- Source code: <https://github.com/dbos-inc/dbos-transact-py>
- Documentation: <https://docs.dbos.dev/>
- Examples: <https://docs.dbos.dev/examples>

# DBOS

# Backup Slides

```
@app.post("/checkout/{idempotency_key}")
def checkout_endpoint(idempotency_key: str) -> Response:
    # Idempotently start the checkout workflow in the background.
    with SetWorkflowID(idempotency_key):
        handle = DBOS.start_workflow(checkout_workflow)
```