Implementing a Reliable Workflow Engine in TypeScript

Peter Kraft

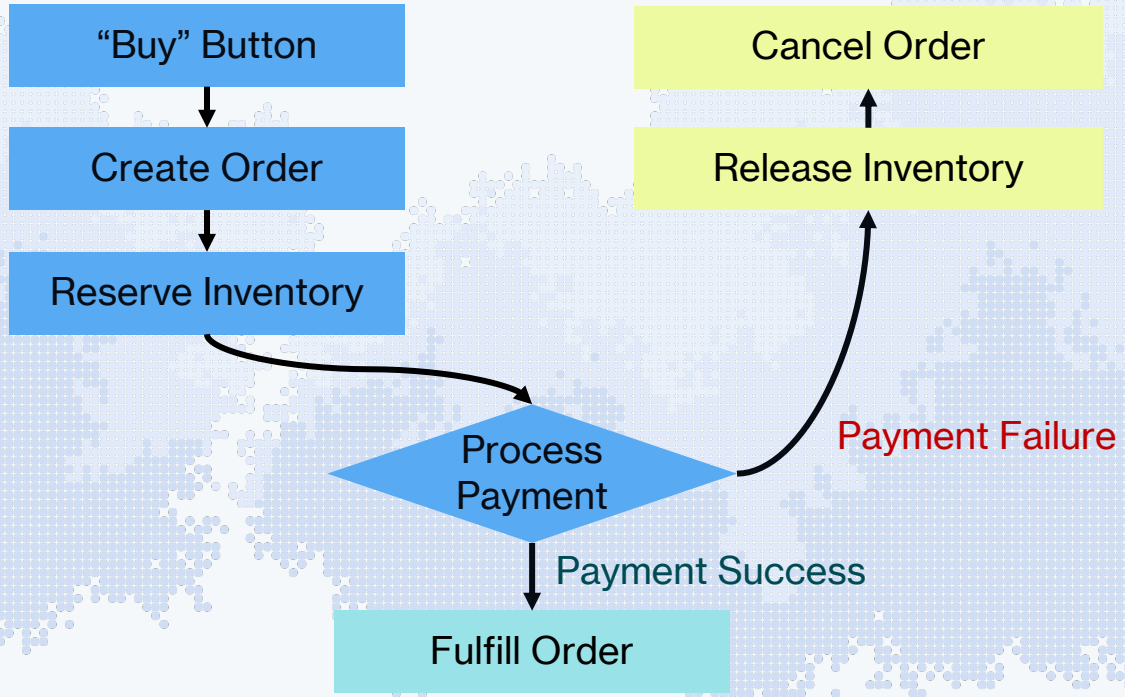
© 2024 dbos.dev

What Are Reliable Workflows?

Programs that, when interrupted, automatically resume from exactly where they left off.

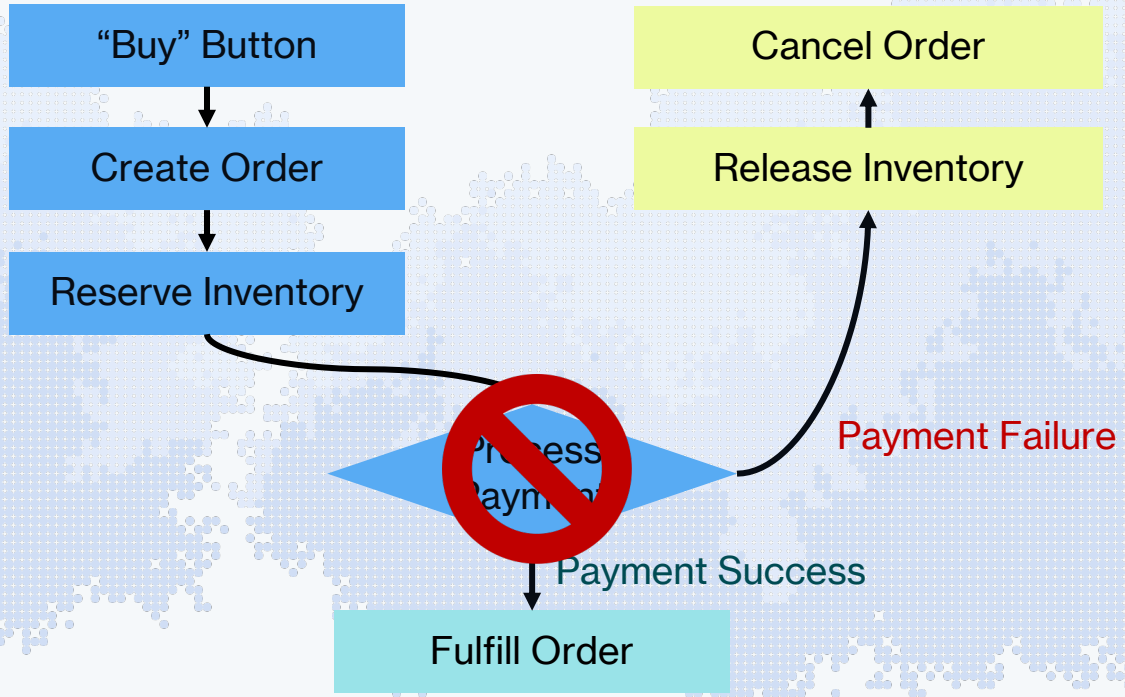
© 2024 dbos.dev

Example Workflow: A “Buy Button”



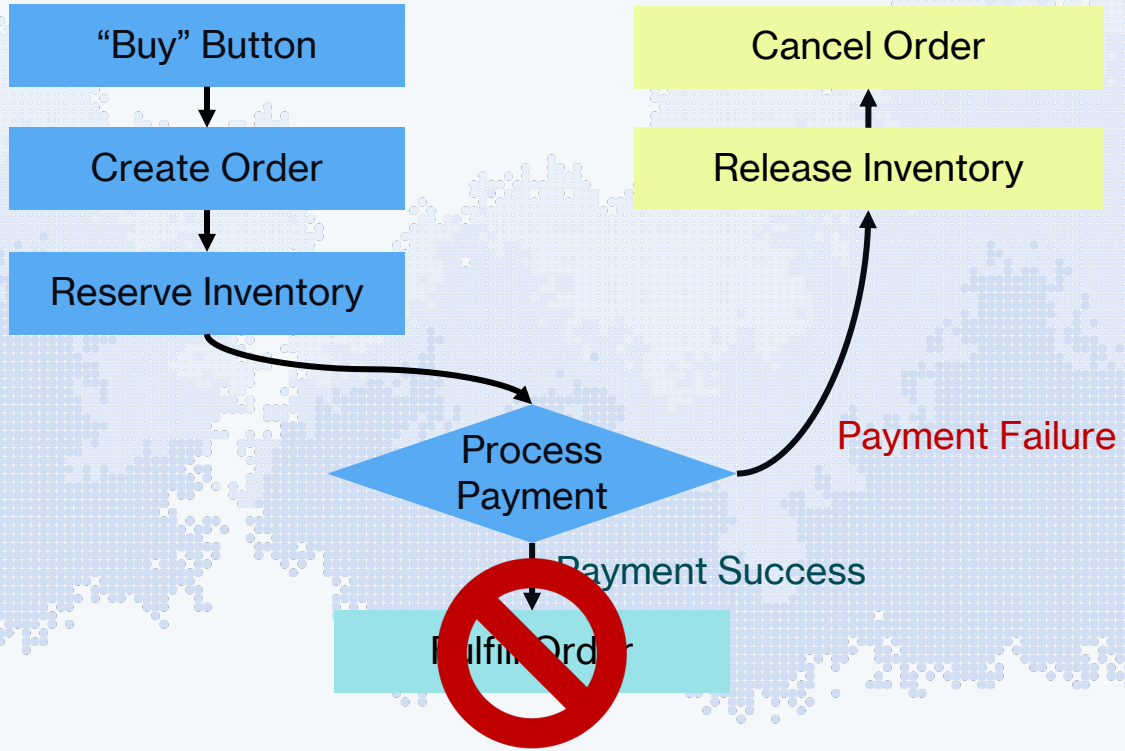
© 2024 dbos.dev

What If You Fail During Payment Processing?



© 2024 dbos.dev

What If You Fail During Order Fulfillment?



© 2024 dbos.dev

How Do You Write Workflows?

	Pros	Cons
Build your own state machine	Performant, no additional dependencies	Brittle and hard to maintain
Use an external orchestrator (e.g.: AWS Step Functions)	Easier to develop with	High performance overhead, adds architectural complexity

© 2024 dbos.dev

How do we get the best of both worlds?

© 2024 dbos.dev

Do it in TypeScript

- Implement workflows at the framework level
- Wrap your operations in code building a state machine
- Store execution state in your app database

https://github.com/dbos-inc/dbos-transact



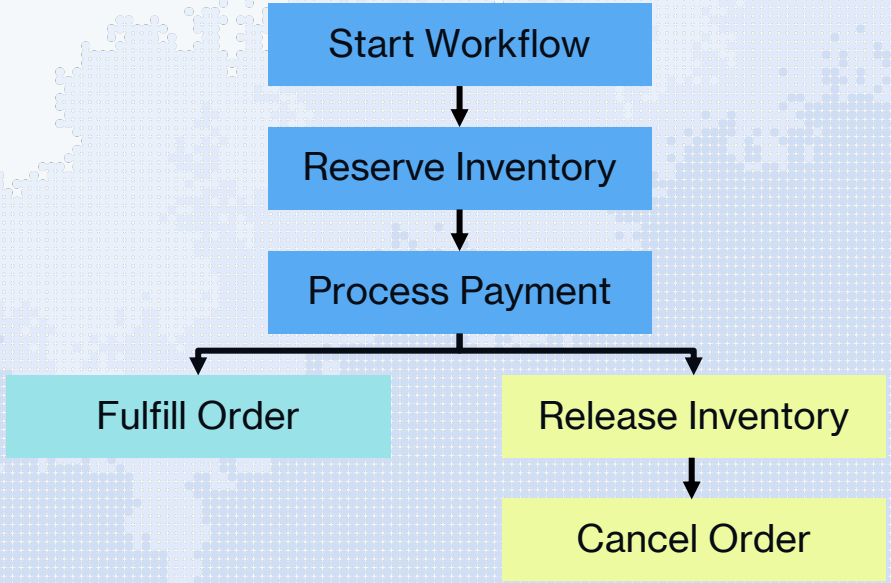
© 2024 dbos.dev

Demo

© 2024 dbos.dev

Write Workflows in Native TypeScript

```
@Workflow()
static async paymentWorkflow() {
  await reserveInventory();
  const success = await processPayment();
  if (success) {
    await fullfillOrder();
  } else {
    await releaseInventory();
    await cancelOrder();
  }
  return success
}
```



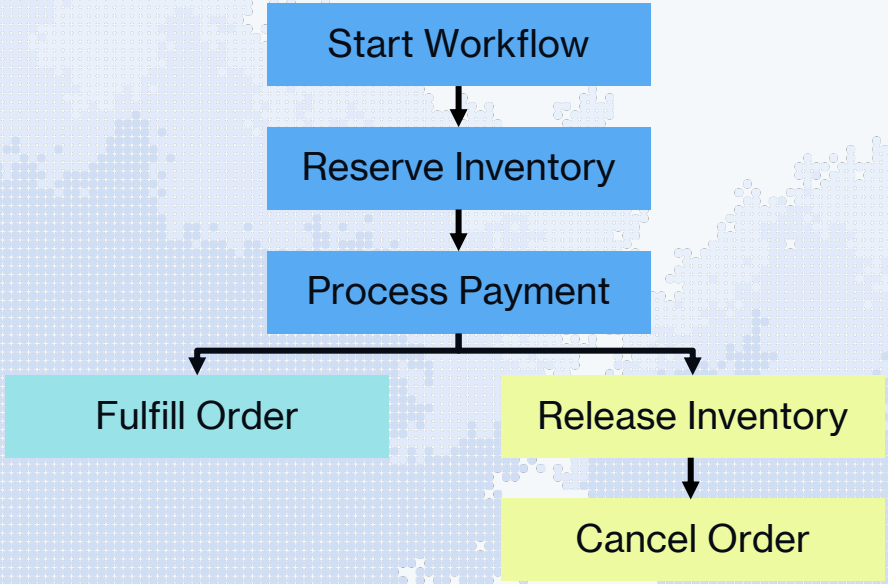
© 2024 dbos.dev

Wrap Functions Using Decorators

`@Workflow()`	Orchestrate a reliable workflow
`@Transaction()`	Call a database transaction
`@Communicator()`	Call an external service or API

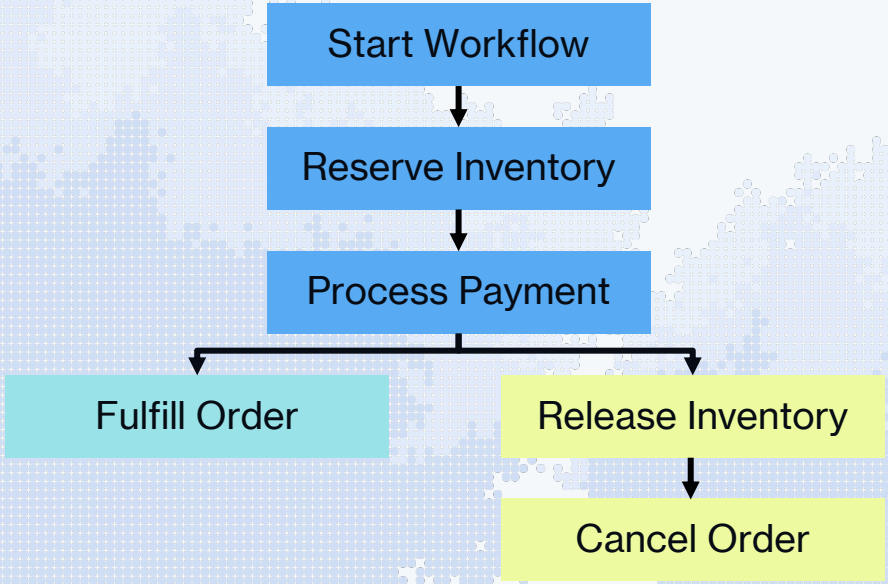
© 2024 dbos.dev

How It Works: Build a State Machine



© 2024 dbos.dev

Record the Workflow Start

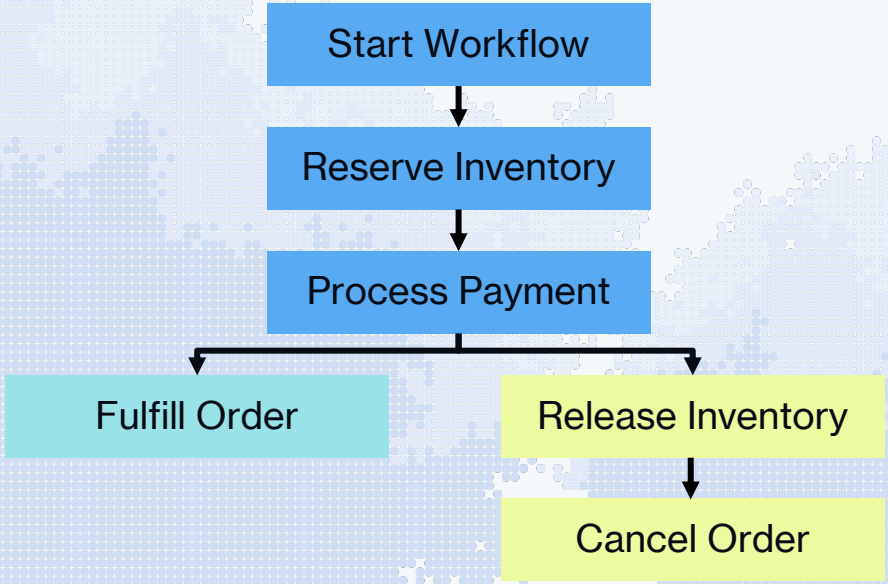


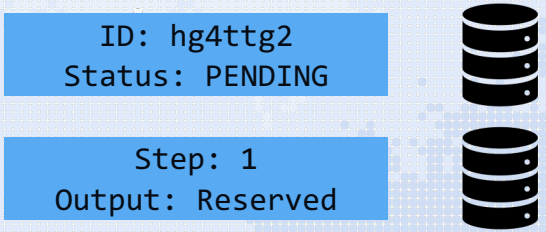

```
ID: hg4ttg2
Status: PENDING
```



© 2024 dbos.dev

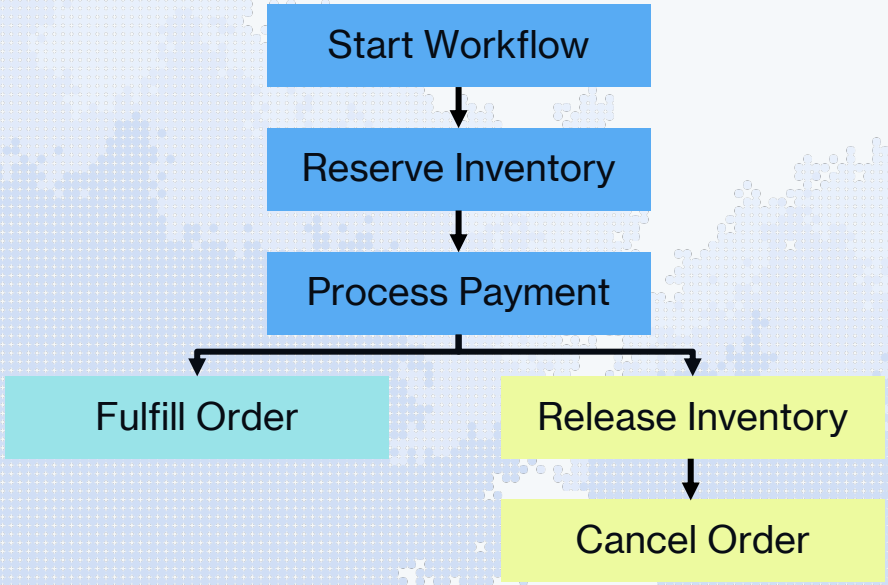
Record Each Step's Output

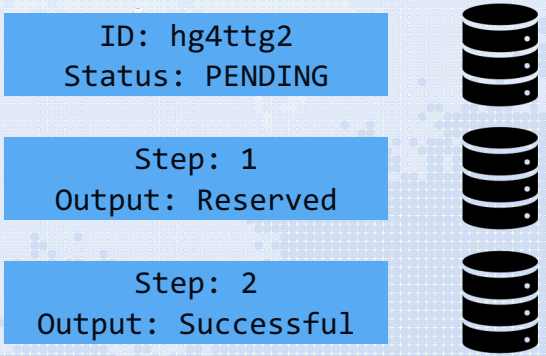




© 2024 dbos.dev

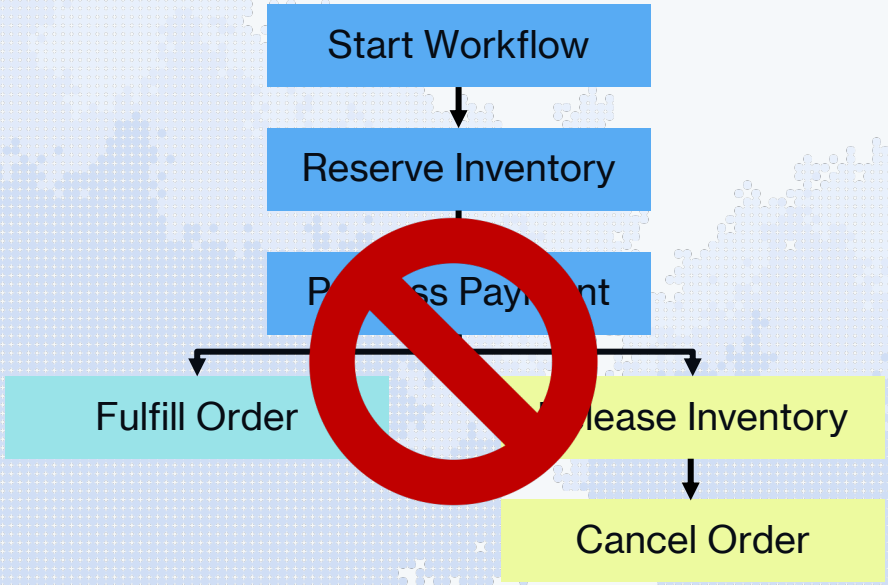
Record Each Step's Output

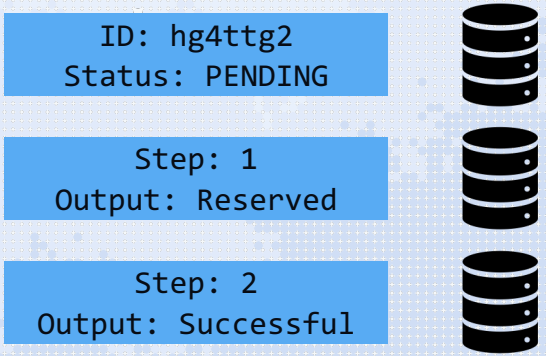




© 2024 dbos.dev

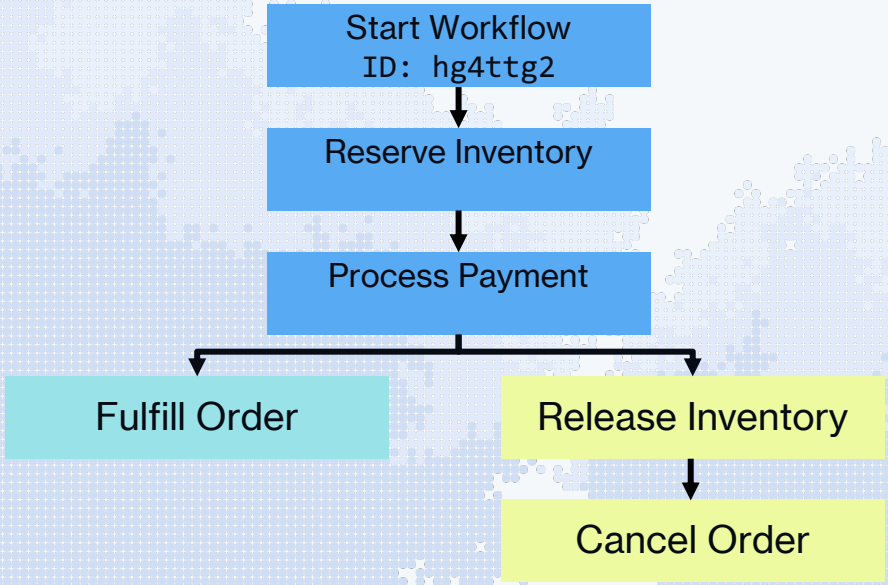
What If It Fails?

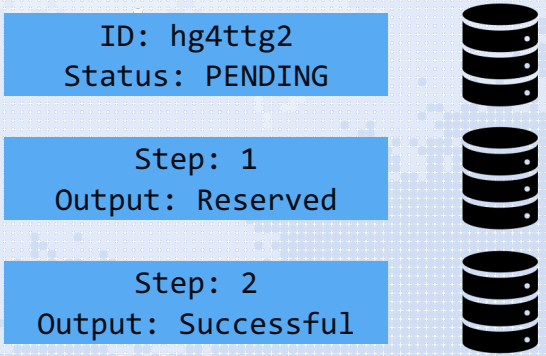




© 2024 dbos.dev

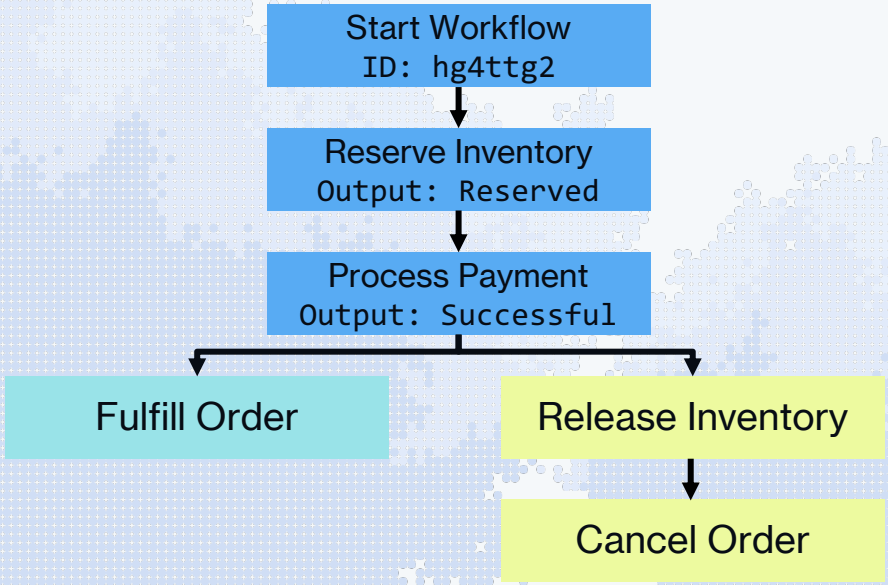
Restart and Check Recorded Outputs

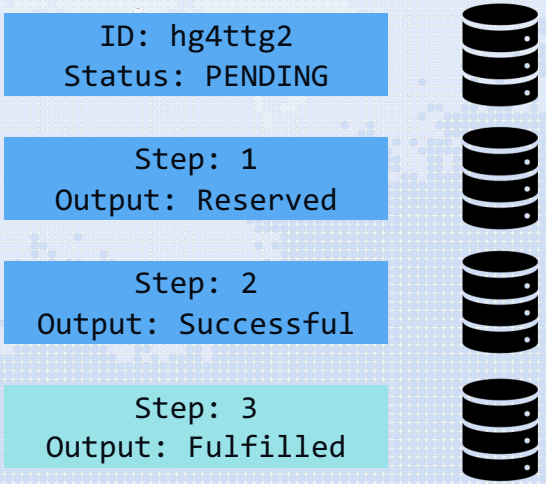




© 2024 dbos.dev

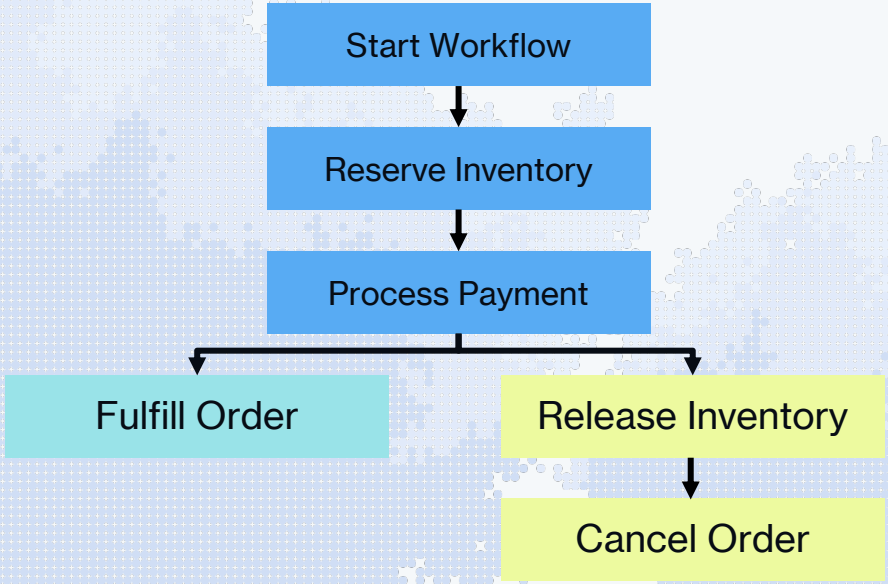
Resume From Where You Left Off





© 2024 dbos.dev

Resume From Where You Left Off




```
ID: hg4ttg2
Status: SUCCESSFUL
```



© 2024 dbos.dev

How To Resume From Where You Left Off

- Before starting a workflow, record its ID in the database
- After executing an operation, record its output in the database
- On restart, re-execute all PENDING workflows
- During re-execution, skip operations with recorded outputs

© 2024 dbos.dev

Aside: Why TypeScript?

TypeScript Feature	Pros
First-Class Functions + Decorators	Makes wrapping functions easy and natural
Robust Type System	Wrapping functions preserves their type
Async/Await	You can start a workflow without waiting for it to complete -> great for event processing
Node.js Event Loop	Highly resource-efficient

© 2024 dbos.dev

Thank you!

- Write reliable workflows in native TypeScript
- Use decorators to wrap functions
- TypeScript is a great language for database applications!

https://github.com/dbos-inc/dbos-transact



© 2024 dbos.dev