**Suggestions**

1. Separate the file contents from JSON payload. Client applications can upload the files and provide pointers/identifiers as part of payload. This will reduce the performance issue and any other issues due to high volume of payload.
2. The API should store the payload first with (Document store or a temp db) as part of transaction and then send acknowledge with client. Currently GUID is returned while the payload is saved with a separate thread. This will improve error handing and traceability.
3. The queue should be monitored by multiple instances of processor, like competing consumer’s pattern. This will improve the performance and scalability.
4. The number of instances of processor can be scaled based on number of messages in the queue. This will ensure better utilization of resources. Not sure how this can be achieved using current framework.
5. The processing job should be divided into multiple independent executable units, which can be scaled, deployed and monitored independently. The steps can be reused to process various kinds of orders.
6. The long running processing job can be designed as process manager / saga (Refer NServiceBus Saga).
7. With process manager, steps can be monitored, and retried appropriately. This will provide a detailed status into request processing and manual intervention as required.
8. The multiple steps can communicate using messages. For example, for a given payload the process manager will decide required processing steps.

Like

1. Create an order in order central.
2. Create 10 order items in order central.
3. Upload 20 documents to document store.
4. Link documents to order once documents are created
5. Create an order (?) with verifile (?).
6. Link verifile and order central orders together

This approach can provide a better flexibility to change the processing logic in future. The process manager can raise events/commands which will be subscribed by various listeners (steps) to take required action. Event based communication will ensure there is no direct dependency between process manager and processing steps. The processing steps will be configurable for each type of order.

**Suggested implementation options**

* Azure Service Fabric for implementing processing steps (Service Fabric can be deployed to on-premises servers)
* Sagas (NServiceBus) for Process Manager

**Using UI Composition**

* UI composition can help to stich data from verifile and order central to provide a unified view of the order.
* The processing job can create order, order items individually in verifile and order central, and provide a tracking number to client. The mapping between order ids across systems and tracking number needs to be maintained in some form.
* Existing UI application can be changed to show indication that an order from order central/verifile is part of bundle.
* The tracking number can be used to fetch data from verifile and order central. This will minimize any code changes to backend code for these two systems.