Techpuzzle 7 Solution
rloesch@

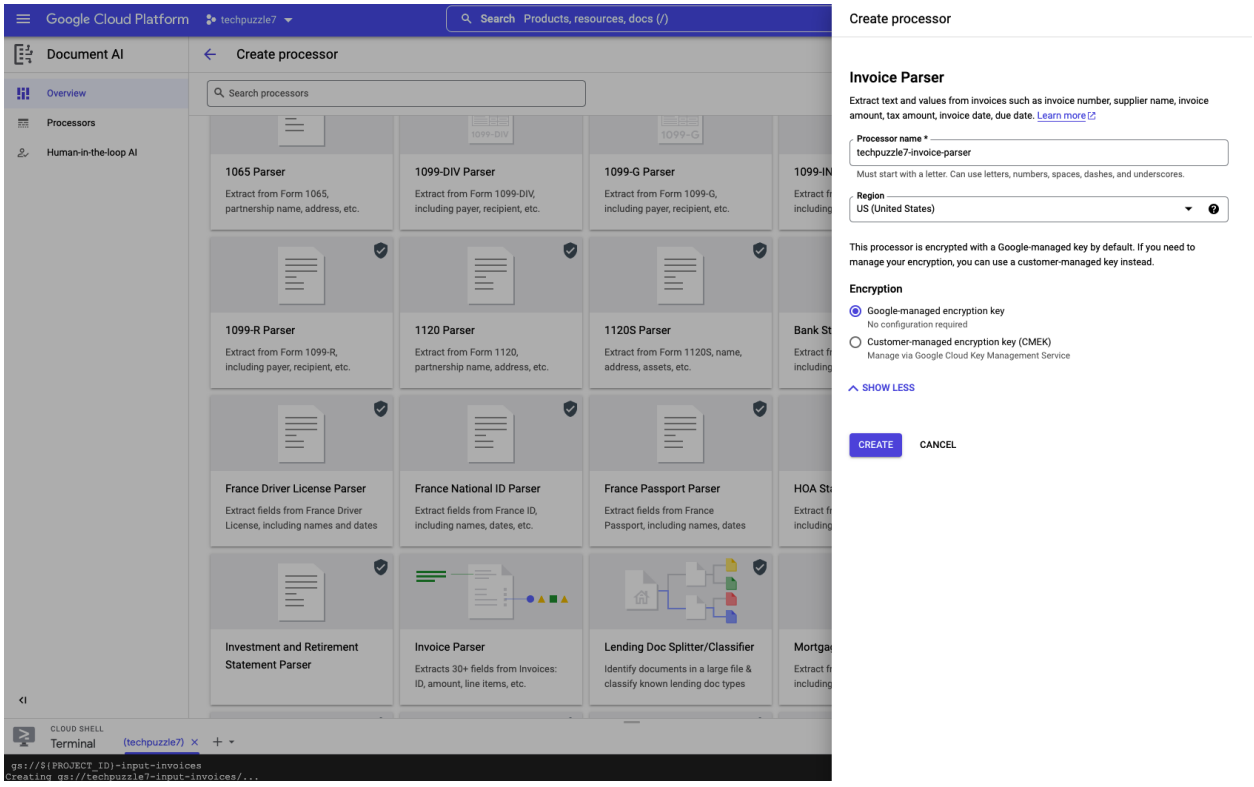
Enable the APIs required

```
gcloud services enable documentai.googleapis.com
gcloud services enable cloudfunctions.googleapis.com
….
```

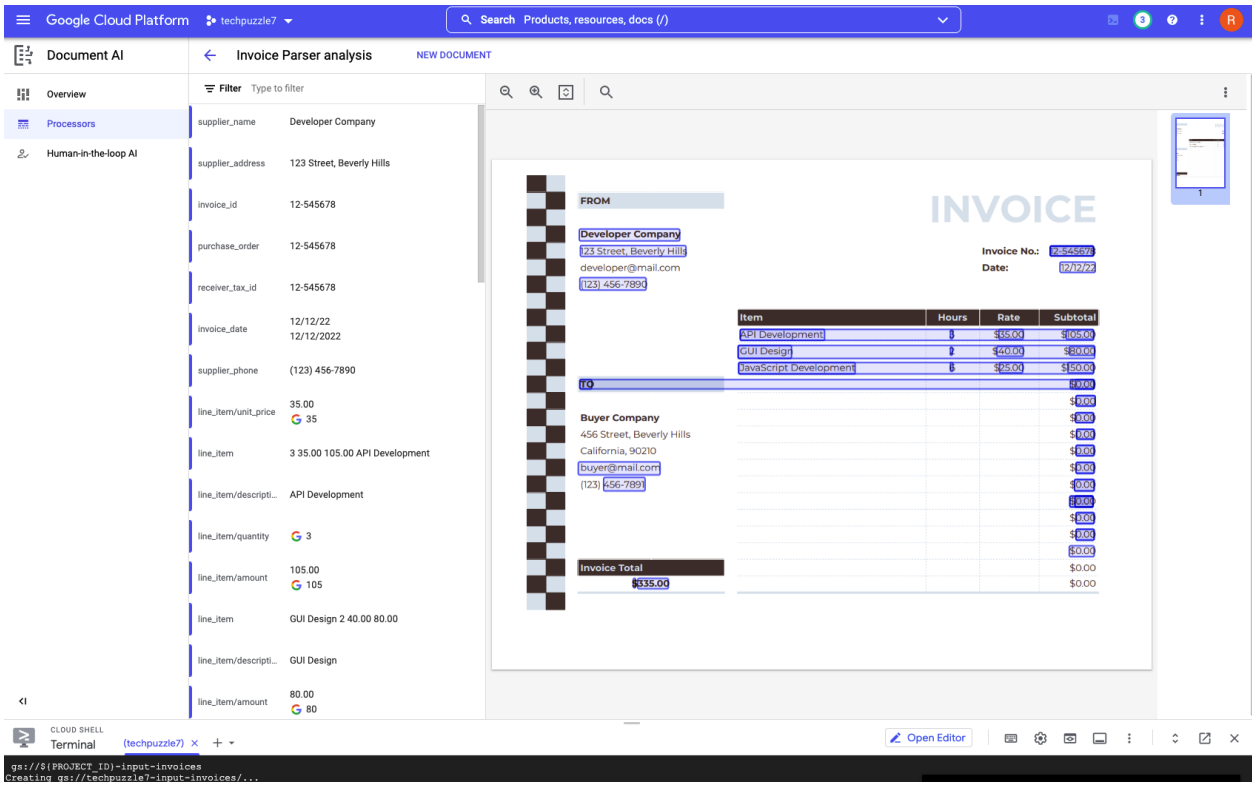
Create Cloud Storage buckets

```
export PROJECT_ID=techpuzzle7
export BUCKET_LOCATION=us-central1
gsutil mb -c standard -l ${BUCKET_LOCATION} -b on \
 gs://${PROJECT_ID}-invoices
```

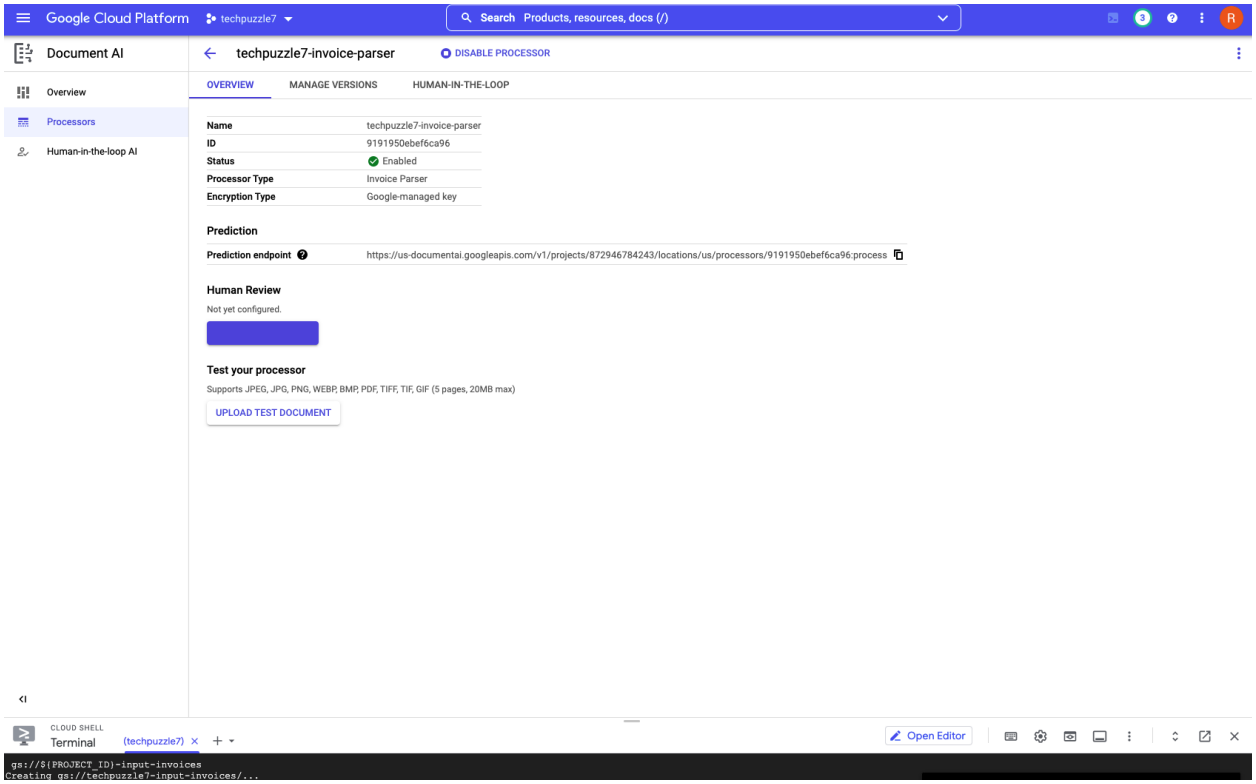
Create the Invoice Parser



Test Invoice Parser

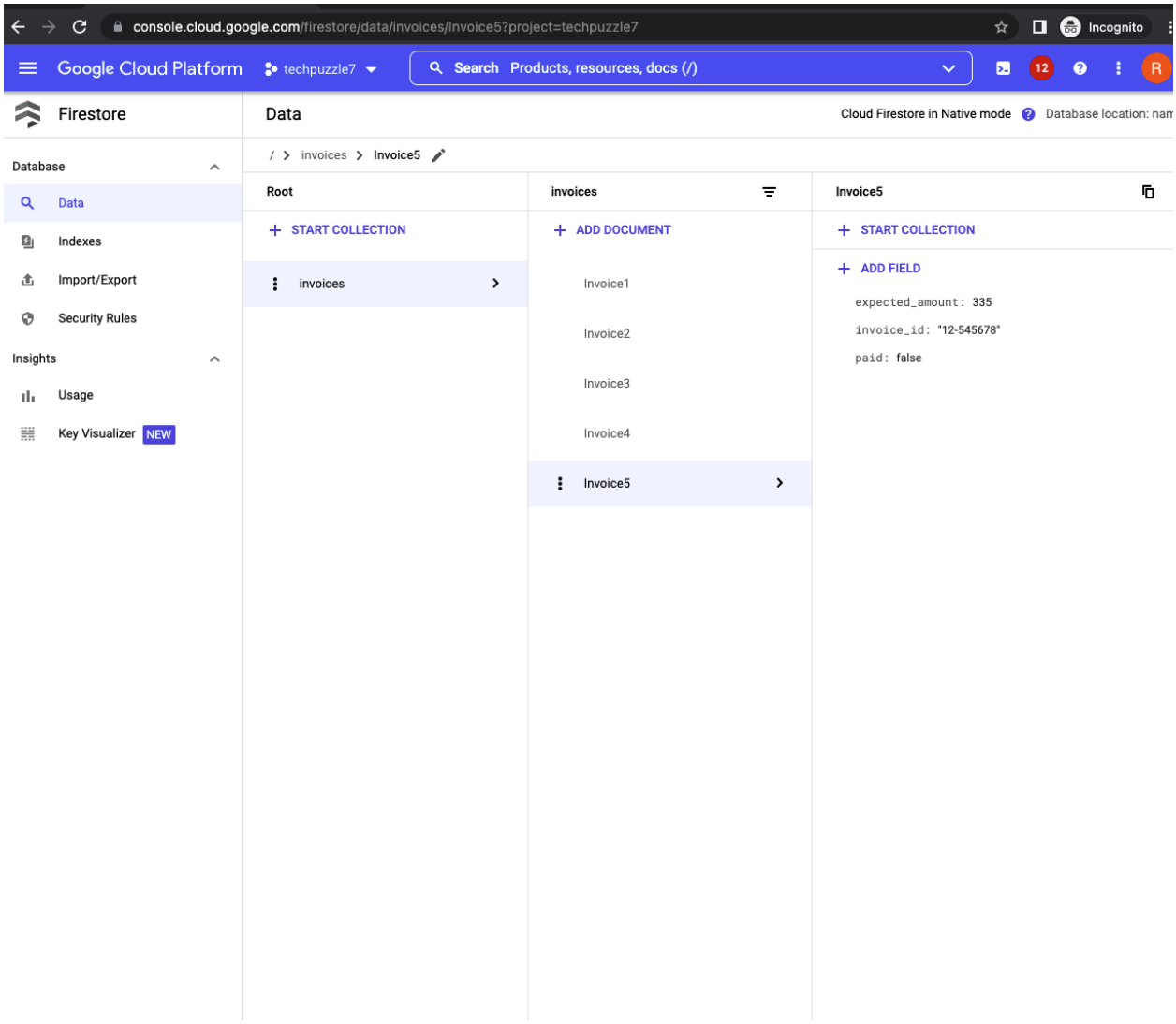


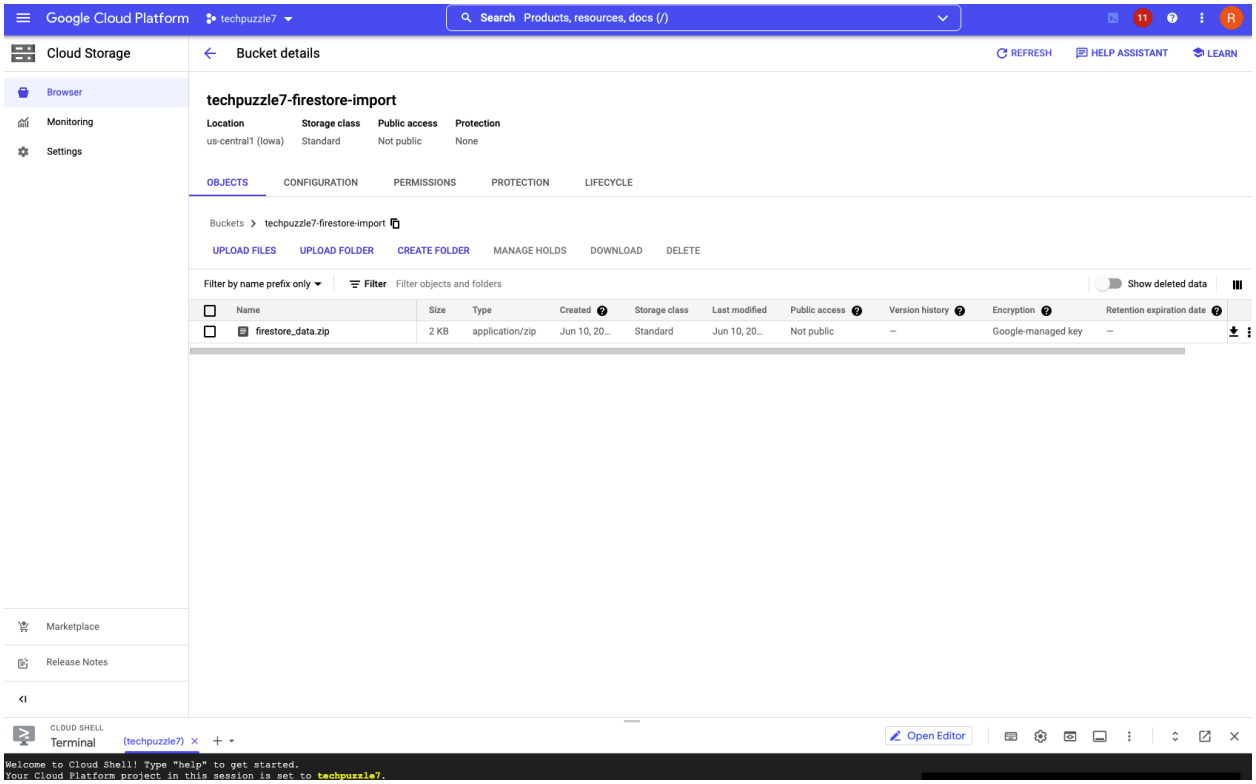
Review Completed Invoice Parser



Create Firestore and import the provided data

gcloud firestore import gs://techpuzzle7-firestore-import/2022-05-27T15:15:39_80829


console.cloud.google.com/firestore/data/invoices/Invoice5?project=techpuzzle7
Incognito
Google Cloud Platform
techpuzzle7
Search Products, resources, docs (/)
12
R
Firestore
Data
Cloud Firestore in Native mode
Database location: nam
Database
Data
Indexes
Import/Export
Security Rules
Insights
Usage
Key Visualizer
NEW
/ > invoices > Invoice5
Root
START COLLECTION
invoices
invoices
ADD DOCUMENT
Invoice1
Invoice2
Invoice3
Invoice4
Invoice5
Invoice5
START COLLECTION
ADD FIELD
expected_amount: 335
invoice_id: "12-545678"
paid: false



Write Cloud Function

```
package techpuzzle7

import (
   "context"
   "fmt"
   "io/ioutil"
   "math"
   "net/http"
   "reflect"
   "strconv"
   "strings"
   "time"

   documentai "cloud.google.com/go/documentai/apiv1"
   "cloud.google.com/go/firestore"
   "cloud.google.com/go/storage"
   "google.golang.org/api/iterator"
```

```
    documentaipb "google.golang.org/genproto/googleapis/cloud/documentai/v1"
)

var (
    storageClient   *storage.Client
    firestoreClient *firestore.Client
)

const processor string =
"projects/872946784243/locations/us/processors/9191950ebef6ca96"
const projectID string = "techpuzzle7"

// GCSEvent is the payload of a GCS event.
type GCSEvent struct {
    Bucket         string    `json:"bucket"`
    Name           string    `json:"name"`
    Metageneration string    `json:"metageneration"`
    ResourceState  string    `json:"resourceState"`
    TimeCreated    time.Time `json:"timeCreated"`
    Updated        time.Time `json:"updated"`
}

type Invoice struct {
    Invoice_id string
    Amount     float64
    Paid       bool
}

// ProcessInvoice is executed when a file is uploaded to the Cloud Storage bucket you
// created for uploading invoices. It processes the invoice for text and updates the
// invoice data in Firebase.
func ProcessInvoice(ctx context.Context, event GCSEvent) error {

    // Read the file from GCS that triggered the event and get it as a byte array
    if event.Bucket == "" {
        return fmt.Errorf("empty file.Bucket")
    }
    if event.Name == "" {
        return fmt.Errorf("empty file.Name")
    }

    storageClient, err := storage.NewClient(ctx)
```

```
    if err != nil {
        return fmt.Errorf("failed to create client: %v", err)
    }
    bucket := storageClient.Bucket(event.Bucket)

    rc, err := bucket.Object(event.Name).NewReader(ctx)
    if err != nil {
        return fmt.Errorf("readFile: unable to open file from bucket %q, file %q: %v",
event.Bucket, event.Name, err)
    }
    defer rc.Close()
    data, err := ioutil.ReadAll(rc)
    if err != nil {
        return fmt.Errorf("readFile: unable to read data from bucket %q, file %q: %v",
event.Bucket, event.Name, err)
    }

    //Detect the mime type of the file
    mimeType := http.DetectContentType(data)

    //Invoke the document processor
    client, error := documentai.NewDocumentProcessorClient(ctx)
    if error != nil {
        return fmt.Errorf("failed to create new document processor client: %v", error)
    }
    defer client.Close()
    req := &documentaipb.ProcessRequest{
        Source: &documentaipb.ProcessRequest_RawDocument{
            RawDocument: &documentaipb.RawDocument{
                Content:  data,
                MimeType: mimeType,
            },
        },
        Name: processor,
    }
    resp, err := client.ProcessDocument(ctx, req)
    if err != nil {
        return fmt.Errorf("failed to process response: %v", err)
    }

    // Parse through the detected fields on the invoice
    document := resp.GetDocument()
```

```
    entities := document.GetEntities()
    var inv Invoice
    for _, entity := range entities {
        if entity.GetType() == "invoice_id" {
            inv.Invoice_id = entity.GetMentionText()
        }
        if entity.GetType() == "total_amount" {
            var strAmount string = entity.GetMentionText()
            strAmount = strings.Replace(strAmount, ",", "", -1)
            fAmount, err := getFloat(strAmount)
            if err == nil {
                inv.Amount = fAmount
            } else {
                return fmt.Errorf("failed to convert amount: %v", err)
            }
        }
    }
    fmt.Printf("Invoice ID: %s\n", inv.Invoice_id)
    fmt.Printf("Total Amount: %.2f\n", inv.Amount)

    // Get a Firestore client.
    firestoreClient, err = firestore.NewClient(ctx, projectID)
    if err != nil {
        return fmt.Errorf("failed to create firestore.NewClient: %v", err)
    }
    defer firestoreClient.Close()

    //Look for the invoice in Firestore
    iter := firestoreClient.Collection("invoices").Where("invoice_id", "==",
inv.Invoice_id).Documents(ctx)

    for {
        dsnap, err := iter.Next()
        if err == iterator.Done {
            break
        }
        if err != nil {
            fmt.Printf("Invoice %s does not exist.\n", inv.Invoice_id)
            return fmt.Errorf("failed to convert amount: %v", err)
        }
```

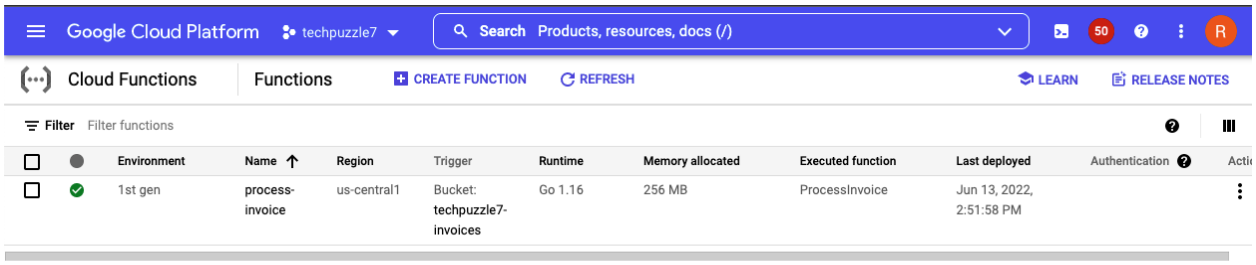
```
        // Taking care of cases where the invoice is already paid or the amount doesn't
match the expected amount
        invoiceMap := dsnap.Data()
        var invData Invoice
        invData.Invoice_id = invoiceMap["invoice_id"].(string)
        iAmount := invoiceMap["expected_amount"]
        fAmount, err := getFloat(iAmount)
        if err == nil {
            inv.Amount = fAmount
        } else {
            return fmt.Errorf("failed to convert amount: %v", err)
        }

        invData.Amount = fAmount
        invData.Paid = invoiceMap["paid"].(bool)
        strPaid := "unpaid"
        if invData.Paid {
            strPaid = "paid"
            fmt.Printf("The invoice %s has already been paid!\n", invData.Invoice_id)
            // TODO: Implement duplicate invoice logic
        }
        fmt.Printf("Invoice %s has an expected amount of %.2f and is in status %s.\n",
invData.Invoice_id, invData.Amount, strPaid)

        if inv.Amount != invData.Amount {
            fmt.Printf("The paid amount %.2f does not match the expected amount
%.2f.\n", inv.Amount, invData.Amount)
            // TODO: Implement unexpected amount logic
        }

        // Update the Firestore record
        _, err = dsnap.Ref.Update(ctx, []firestore.Update{
            {
                Path:  "paid",
                Value: true,
            }})

        if err != nil {
            return fmt.Errorf("failed to update invoice: %v", err)
        }

    }
```

```
    return nil
}

// Helper function to convert amount to float where possible
var floatType = reflect.TypeOf(float64(0))
var stringType = reflect.TypeOf("")

func getFloat(unk interface{}) (float64, error) {
    switch i := unk.(type) {
    case float64:
        return i, nil
    case float32:
        return float64(i), nil
    case int64:
        return float64(i), nil
    case int32:
        return float64(i), nil
    case int:
        return float64(i), nil
    case uint64:
        return float64(i), nil
    case uint32:
        return float64(i), nil
    case uint:
        return float64(i), nil
    case string:
        return strconv.ParseFloat(i, 64)
    default:
        v := reflect.ValueOf(unk)
        v = reflect.Indirect(v)
        if v.Type().ConvertibleTo(floatType) {
            fv := v.Convert(floatType)
            return fv.Float(), nil
        } else if v.Type().ConvertibleTo(stringType) {
            sv := v.Convert(stringType)
            s := sv.String()
            return strconv.ParseFloat(s, 64)
        } else {
            return math.NaN(), fmt.Errorf("Can't convert %v to float64", v.Type())
        }
    }
```

```
}
```

Deploy Cloud Function

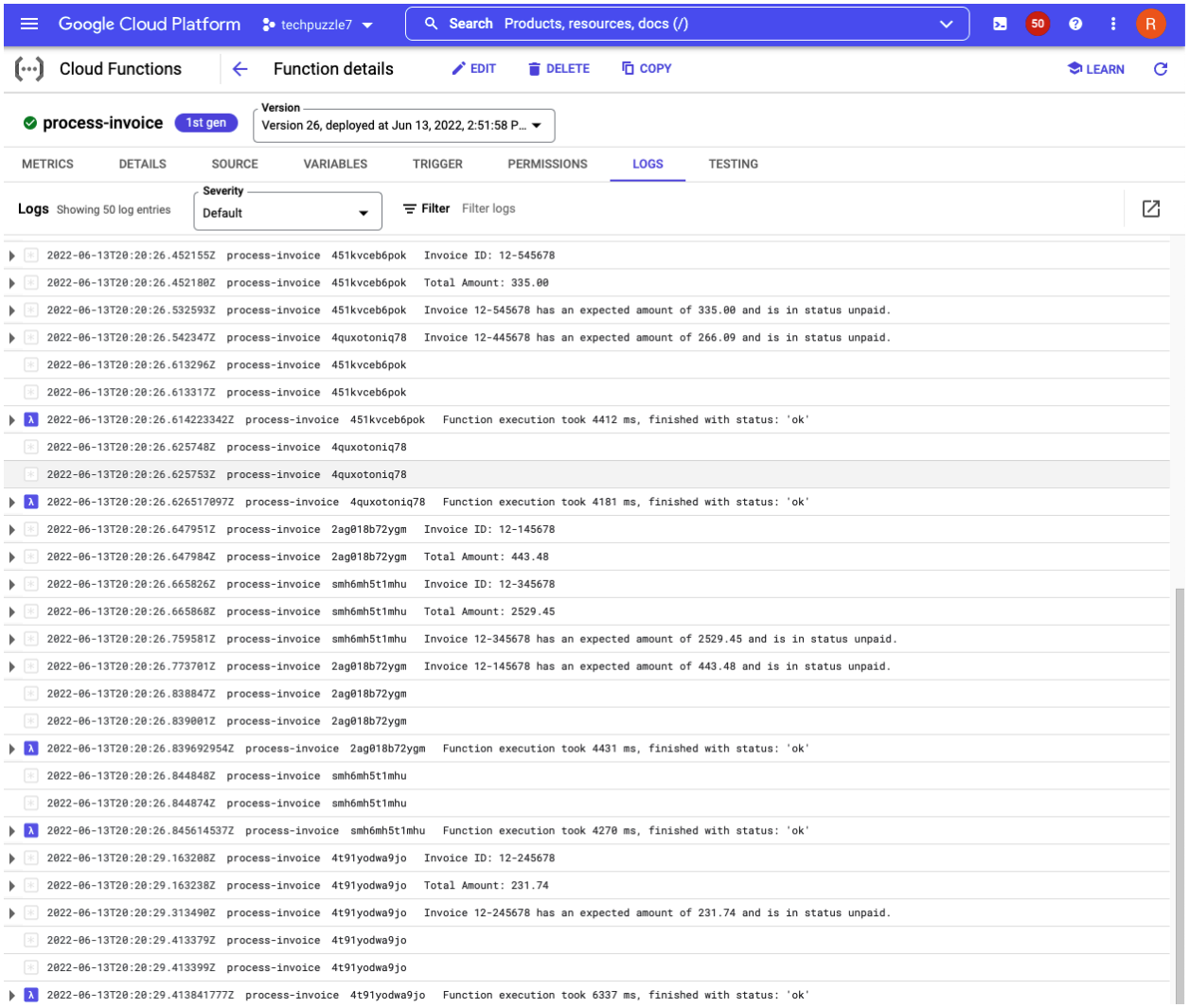
gcloud functions deploy process-invoice --runtime go116 --trigger-bucket techpuzzle7-invoices --entry-point ProcessInvoice

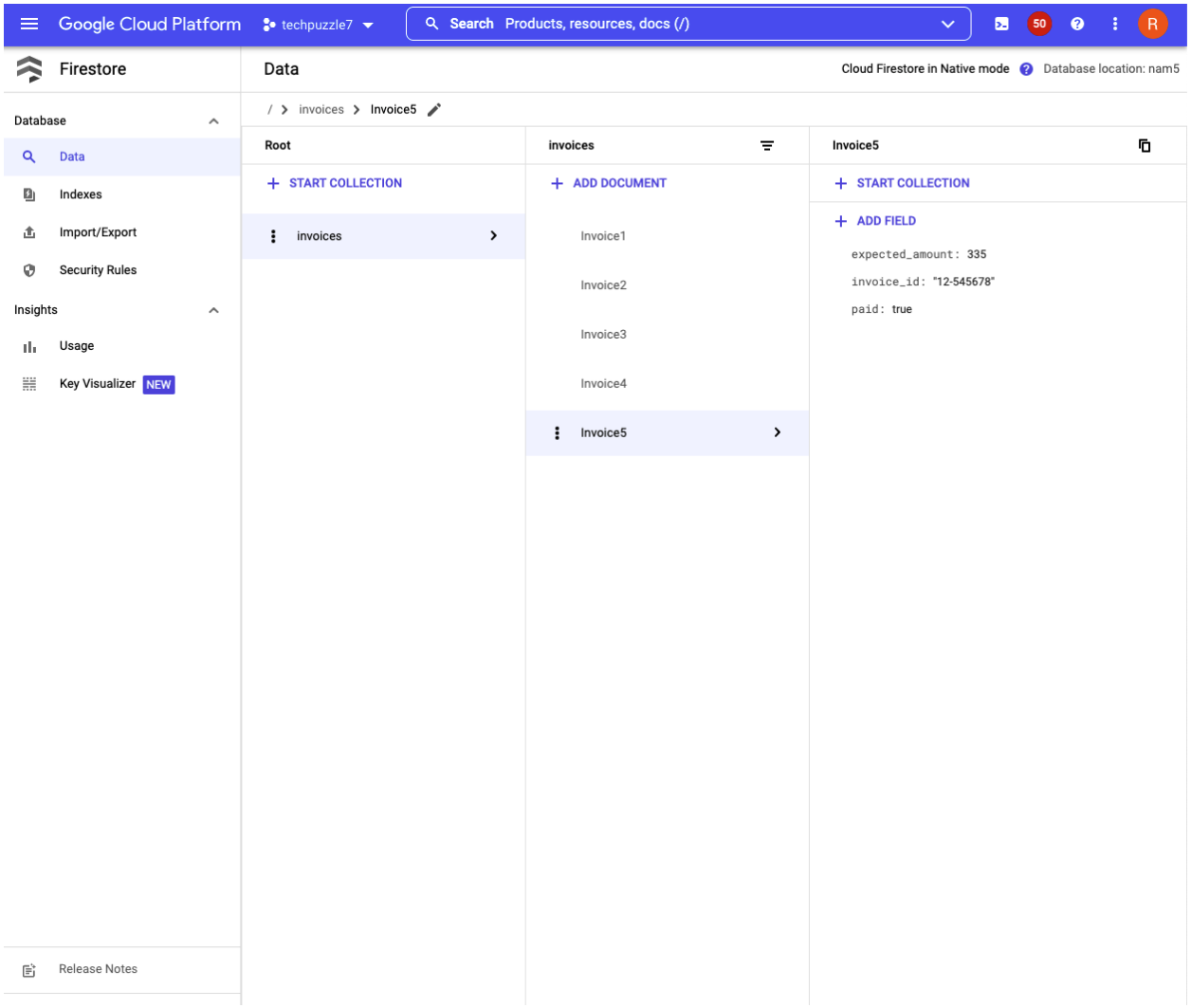


Test File Upload and File Processing

vscode ➜ /workspaces/Go Programming/techpuzzle7 $ ls -l ./samples/
total 956
-rw-r--r-- 1 vscode vscode 27753 Jun 13 15:50 'Sample Invoice 1.pdf'
-rw-r--r-- 1 vscode vscode 127861 Jun 13 15:51 'Sample Invoice 1.png'
-rw-r--r-- 1 vscode vscode 45750 Jun 13 15:50 'Sample Invoice 2.pdf'
-rw-r--r-- 1 vscode vscode 166775 Jun 13 15:51 'Sample Invoice 2.png'
-rw-r--r-- 1 vscode vscode 25725 Jun 13 13:43 'Sample Invoice 3.pdf'
-rw-r--r-- 1 vscode vscode 145461 Jun 13 15:51 'Sample Invoice 3.png'
-rw-r--r-- 1 vscode vscode 118689 Jun 13 15:50 'Sample Invoice 4.pdf'
-rw-r--r-- 1 vscode vscode 147516 Jun 13 15:51 'Sample Invoice 4.png'
-rw-r--r-- 1 vscode vscode 17841 Jun 13 15:51 'Sample Invoice 5.pdf'
-rw-r--r-- 1 vscode vscode 134351 Jun 13 15:51 'Sample Invoice 5.png'
vscode ➜ /workspaces/Go Programming/techpuzzle7 $ gsutil -m cp ./samples/*.pdf gs://techpuzzle7-invoices
Copying file://./samples/Sample Invoice 1.pdf [Content-Type=application/pdf]...
Copying file://./samples/Sample Invoice 2.pdf [Content-Type=application/pdf]...
Copying file://./samples/Sample Invoice 4.pdf [Content-Type=application/pdf]...
Copying file://./samples/Sample Invoice 3.pdf [Content-Type=application/pdf]...

Copying file://./samples/Sample Invoice 5.pdf [Content-Type=application/pdf]...
- [5/5 files][230.2 KiB/230.2 KiB] 100% Done
Operation completed over 5 objects/230.2 KiB.
vscode ➜ /workspaces/Go Programming/techpuzzle7 $





Special Cases

Notes ...

Can you handle the case where a duplicate invoice is received?

What if the payment amount of the invoice doesn't match the expected amount?

What if we receive an invoice and don't know which order to apply it to?

```
for {
    dsnap, err := iter.Next()
    if err == iterator.Done {
        break
    }
```

```
        if err != nil {
            fmt.Printf("Invoice %s does not exist.\n", inv.Invoice_id)
            // TODO: Implement logic if invoice doesn't exist
            return fmt.Errorf("failed to convert amount: %v", err)
        }

        // Taking care of cases where the invoice is already paid or the amount doesn't
match the expected amount
        invoiceMap := dsnap.Data()
        var invData Invoice
        invData.Invoice_id = invoiceMap["invoice_id"].(string)
        iAmount := invoiceMap["expected_amount"]
        fAmount, err := getFloat(iAmount)
        if err == nil {
            inv.Amount = fAmount
        } else {
            return fmt.Errorf("failed to convert amount: %v", err)
        }

        invData.Amount = fAmount
        invData.Paid = invoiceMap["paid"].(bool)
        strPaid := "unpaid"
        if invData.Paid {
            strPaid = "paid"
            fmt.Printf("The invoice %s has already been paid!\n", invData.Invoice_id)
            // TODO: Implement duplicate invoice logic
        }
        fmt.Printf("Invoice %s has an expected amount of %.2f and is in status %s.\n",
invData.Invoice_id, invData.Amount, strPaid)

        if inv.Amount != invData.Amount {
            fmt.Printf("The paid amount %.2f does not match the expected amount
%.2f.\n", inv.Amount, invData.Amount)
            // TODO: Implement unexpected amount logic
        }
```

Useful Links

Package cloud.google.com/go/documentai/apiv1 (v1.4.0)
Go-genproto documentai
Process documents by using client libraries
Invoice Parser
The Go Runtime
Golang-samples
Go googleapi