SiteScanning Developer Handoff

Key Repositories

The README files in the repositories listed below are comprehensive and provide all the necessary information for developers to begin working on these projects.

Repository Name	Description	Link to Repo
site-scanning	Serves as the home to the Site Scanning project. All project issues are managed through this repository.	https://github.com/GSA/site-scanning
site-scanning-documentation	This repository houses all of the documentation related to the Site Scanning project. This is the first place you should look if you are trying to find anything project related.	https://github.com/GSA/site-scanning-documentation
site-scanning-analysis	This repository houses numerous reports. Some of these reports are scheduled daily through actions and some are one-off datasets used by the federal-website-index.	https://github.com/GSA/site-scanning-analysis
federal-website-index	This is the home of the Federal Website Indexer application. The indexer runs daily and builds a list of all public federal government websites.	https://github.com/GSA/federal-website-index
site-scanning-snapshots	This repository stores a variety of one-off scan data.	https://github.com/GSA/site-scanning-snapshots
site-scanning-engine	This repository houses both the Site Scanning Engine and the API that is used to retrieve the resulting data from the Site Scanning Engine.	https://github.com/GSA/site-scanning-engine

Notes

Among the repositories listed above, developers should pay particular attention to the **site-scanning-engine**, **federal-website-index**, and **site-scanning-analysis** repositories. Each of these includes documentation on how to set up a local development environment to run the projects locally.

Scheduled Workflows

This section outlines the GitHub Actions implemented to ensure the project runs smoothly and on autopilot. The actions are categorized by repository, with descriptions, schedules, and links for each.

site-scanning-engine

Action Name	Description	Schedule	Link to Action
Restart scan worker/consumer	Restarts the scan consumer in cloud.gov to prepare for scan	Runs daily at 00:00 UTC	Link to Action: site-scanning-engine/restart-worker.yml
Enqueue Scans	Queues up the scans loaded from the ingest workflow	Runs at 00:00 UTC on Mon / Wed / Fri	Link to Action: site-scanning-engine/enqueue-scans.yml
Create S3 snapshot	Generates the legacy version of the s3 snapshots	Runs daily at 12:00 UTC	Link to Action: site-scanning-engine/create-snapshot.yml
Requeue stale scans	Queues up the urls that have not had scan data updated for over 3 days.	Runs at 12:00 UTC on Tue / Thur / Sat	Link to Action: site-scanning-engine/requeue-stale-scans.yml
Create daily S3 snapshots	Generates the snapshots located in S3 and archives older versions	Runs daily at 12:15 UTC	Link to Action: site-scanning-engine/create-daily-snapshots.yml

Action Name	Description	Schedule	Link to Action
Create a11y S3 snapshot	Generates the a11y snapshots located in S3	Runs daily at 12:15 UTC	Link to Action: site-scanning-engine/create-a11y-snapshot.yml
Ingest	Ingests all URLs from the federal-site-index	Runs daily at 22:15 UTC	Link to Action: site-scanning-engine/ingest.yml

federal-website-index

Action Name	Description	Schedule	Link to Action
Build DAP Top 100,000 List	Builds the DAP Top 100,00 list that is consumed by the indexer	Runs daily at 21:00 UTC	Link to Action: federal-website-index/build-dap-top-list.yml
Build Final URL List	Builds the final-url list that is consumed by the indexer	Runs daily at 20:30 UTC	Link to Action: federal-website-index/build-finalurl-list.yml
Build target url list using TypeScript	Runs the primary federal website indexer that generates the url list that is consumed by the site-scanning-engine	Runs daily at 20:15 UTC	Link to Action: federal-website-index/build-list-js.yml

site-scanning-analysis

Action Name	Description	Schedule	Link to Action
Generate all reports	Runs all reports within the analysis repo that are not individually scheduled	Runs daily at 13:00 UTC	Link to Action: site-scanning-analysis/generate-all-reports.yml
Generate IDEA Reports	Runs the IDEA Reports	Runs at 00:00 UTC on Thursdays	Link to Action: site-scanning-analysis/generate-idea-reports.yml
Generate Unique Website List	Runs the unique-website-list report	Runs daily at 14:00 UTC	Link to Action: site-scanning-analysis/generate-unique-website-list.yml
Generate URLs Missing From Snapshot	Generates a report of URLs that exist in the index file, but do not have an entry in the daily snapshot	Runs daily at 21:00 UTC	Link to Action: site-scanning-analysis/generate-urls-missing-from-snapshot.yml
Run Smoke Tests	Runs the smoke tests	Runs daily at 13:30 UTC	Link to Action: site-scanning-analysis/smoke-tests.yml

Site Scanning Engine/API: Development and Deployment Overview

The **README** file in the Site Scanning Engine repository provides clear and thorough instructions for running both the scan engine and the API in a local development environment.

Development Workflow

When you're ready to deploy changes, follow these steps:

- **Create a Pull Request (PR):**
  Open a PR to merge your feature branch into the main branch.

- **Automated Checks:**
  Upon creating the PR, several GitHub Actions workflows will run:
  - **Unit and integration tests** to ensure code stability.
  - **Test deployment**, which deploys the scan engine and API to the **development environment** to verify that the deployment process completes successfully.
- **Merge to Main:**
  Once all checks pass, you can safely merge your branch into **main**.
- **Production Deployment:**
  After merging, manually trigger the **"Deploy" GitHub Action** to deploy the scan engine and API to the **production environment**.

Logging and Monitoring

The **Scan Engine** generates extensive logging data, which can be accessed via logs.fr.cloud.gov. All logs are stored in **OpenSearch**, providing powerful search and filtering capabilities.

A number of saved searches are available to help streamline log analysis. These saved queries are all prefixed with **sse** for easy identification.

Useful Links

Name	Description	Link
Access the API	Full instructions on how the we can access the API in the production environment	https://open.gsa.gov/api/site-scanning-api/
Access the Data	Outlines the numerous ways the data can be accessed	https://digital.gov/guides/site-scanning/data/

Name	Description	Link
Architecture Documentation	Contains a diagram that outlines the infrastructure of the scan-engine application	https://github.com/GSA/site-scanning-engine/blob/main/docs/architecture/README.md
Data Dictionary	Defines each field within the scan-engine snapshot	https://github.com/GSA/site-scanning-documentation/blob/main/data/Site_Scanning_Data_Dictionary.csv
How the Federal Website Index is created	This lays out the steps involved in building the Federal Website Index	https://github.com/GSA/federal-website-index/blob/main/process/index-creation.md
How the Scans Run	This lays out the different scans the site-scanning-engine runs along with where they are located	https://github.com/gsa/site-scanning-documentation?tab=readme-ov-file#understanding-the-data
Scan statuses explained	Explains the different scan statuses that returned from the scan-engine	https://github.com/GSA/site-scanning-documentation/blob/main/pages/scan_statuses.md
Site Scanning Issues Board	This is the main board for all issues for the Site Scanning project	https://github.com/GSA/site-scanning/issues
Technical Details	If in doubt, look here.	https://digital.gov/guides/site-scanning/technical-details

CLI Commands

Production Database Tunnel

In order to connect to the production database you need to open a tunnel via CloudFoundry. Once you create the tunnel it will show you the login information needed to connect. Here is the command:

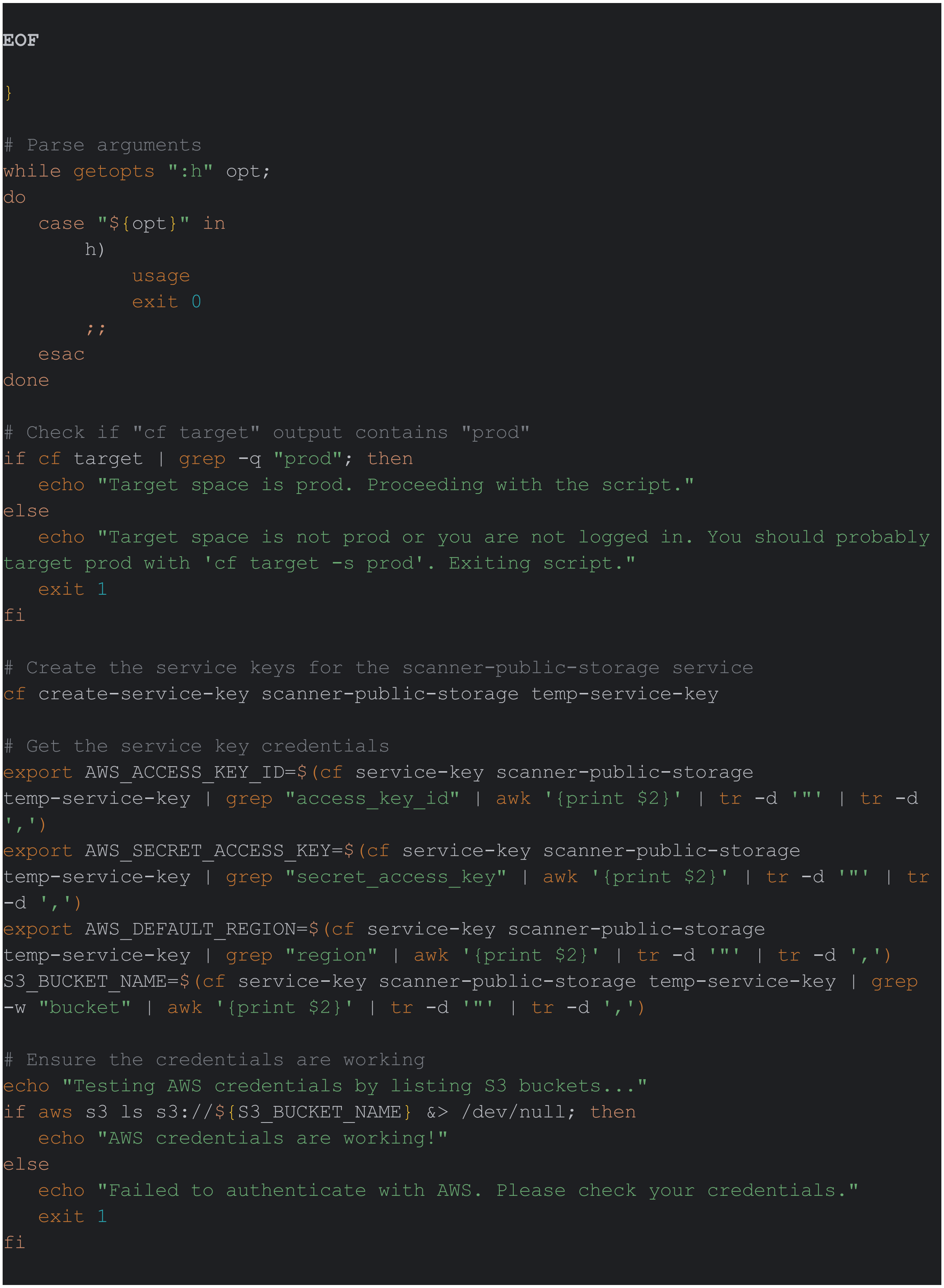
```
cf connect-to-service -no-client site-scanner-consumer
scanner-postgres-02
```

Scripts

I developed a few scripts that I used to do common operations. I never got around to formalizing them for sharing, but a developer might still find them useful, even if only as a reference to create something better.

list-s3-data.sh

```
#!/bin/bash -e

# Check to ensure everything we need to run the script is installed
if ! command -v cf &> /dev/null; then
   echo "cf CLI is not installed. Please install it and try again."
   exit 1
fi

# Define the usage info for the script
function usage () {

cat << EOF
$(basename $(readlink -f "$0")) [OPTIONS]

OPTIONS:
   -h  Show this help message

Description:
Creates temporary AWS keys and list the contents of our S3 bucket
```



```
EOF

}

# Parse arguments
while getopts ":h" opt;
do
   case "${opt}" in
       h)
           usage
           exit 0
       ;;
   esac
done

# Check if "cf target" output contains "prod"
if cf target | grep -q "prod"; then
   echo "Target space is prod. Proceeding with the script."
else
   echo "Target space is not prod or you are not logged in. You should probably target prod with 'cf target -s prod'. Exiting script."
   exit 1
fi

# Create the service keys for the scanner-public-storage service
cf create-service-key scanner-public-storage temp-service-key

# Get the service key credentials
export AWS_ACCESS_KEY_ID=$(cf service-key scanner-public-storage temp-service-key | grep "access_key_id" | awk '{print $2}' | tr -d '"' | tr -d ',')
export AWS_SECRET_ACCESS_KEY=$(cf service-key scanner-public-storage temp-service-key | grep "secret_access_key" | awk '{print $2}' | tr -d '"' | tr -d ',')
export AWS_DEFAULT_REGION=$(cf service-key scanner-public-storage temp-service-key | grep "region" | awk '{print $2}' | tr -d '"' | tr -d ',')
S3_BUCKET_NAME=$(cf service-key scanner-public-storage temp-service-key | grep -w "bucket" | awk '{print $2}' | tr -d '"' | tr -d ',')

# Ensure the credentials are working
echo "Testing AWS credentials by listing S3 buckets..."
if aws s3 ls s3://${S3_BUCKET_NAME} &> /dev/null; then
   echo "AWS credentials are working!"
else
   echo "Failed to authenticate with AWS. Please check your credentials."
   exit 1
fi
```

```
aws s3 ls s3://${S3_BUCKET_NAME} --recursive
```

get-snapshot.sh

```
#!/bin/bash -e

# Check to ensure everything we need to run the script is installed
if ! command -v cf &> /dev/null; then
   echo "cf CLI is not installed. Please install it and try again."
   exit 1
fi
if ! command -v jq &> /dev/null; then
   echo "jq is not installed. Please install it and try again."
   exit 1
fi

# Define the usage info for the script
function usage () {

cat << EOF
$(basename $(readlink -f "$0")) [OPTIONS]

OPTIONS:
   -h  Show this help message

Description:
Creates temporary AWS keys and pulls down the latest snapshot from the S3
bucket

EOF

}

# Parse arguments

while getopts ":h" opt;
do
   case "${opt}" in
       h)
           usage
           exit 0
       ;;
   esac
done
```

```
# Define our variables and ensure export directory

SAVE_DIR=$(dirname "$(readlink -f "$0")")"/../output"

# Check if the directory exists
if [ ! -d "$SAVE_DIR" ]; then
   # Create the directory
   mkdir -p "$SAVE_DIR"
   echo "Created directory: $SAVE_DIR"
else
   echo "Directory already exists: $SAVE_DIR"
fi

# Check if "cf target" output contains "prod"
if cf target | grep -q "prod"; then
   echo "Target space is prod. Proceeding with the script."
else
   echo "Target space is not prod or you are not logged in. You should probably target prod with 'cf target -s prod'. Exiting script."
   exit 1
fi

# Create the service keys for the scanner-public-storage service
cf create-service-key scanner-public-storage temp-service-key

# Get the service key credentials
export AWS_ACCESS_KEY_ID=$(cf service-key scanner-public-storage temp-service-key | grep "access_key_id" | awk '{print $2}' | tr -d '"' | tr -d ',')
export AWS_SECRET_ACCESS_KEY=$(cf service-key scanner-public-storage temp-service-key | grep "secret_access_key" | awk '{print $2}' | tr -d '"' | tr -d ',')
export AWS_DEFAULT_REGION=$(cf service-key scanner-public-storage temp-service-key | grep "region" | awk '{print $2}' | tr -d '"' | tr -d ',')
S3_BUCKET_NAME=$(cf service-key scanner-public-storage temp-service-key | grep -w "bucket" | awk '{print $2}' | tr -d '"' | tr -d ',')

# Ensure the credentials are working
echo "Testing AWS credentials by listing S3 buckets..."
if aws s3 ls s3://${S3_BUCKET_NAME} &> /dev/null; then
   echo "AWS credentials are working!"
else
   echo "Failed to authenticate with AWS. Please check your credentials."
   exit 1
fi

# Prepare to download the latest snapshot
CURRENT_DATE=$(date +%Y%m%d)
SNAPSHOT_FILE_NAME="weekly-snapshot-all.csv"
```

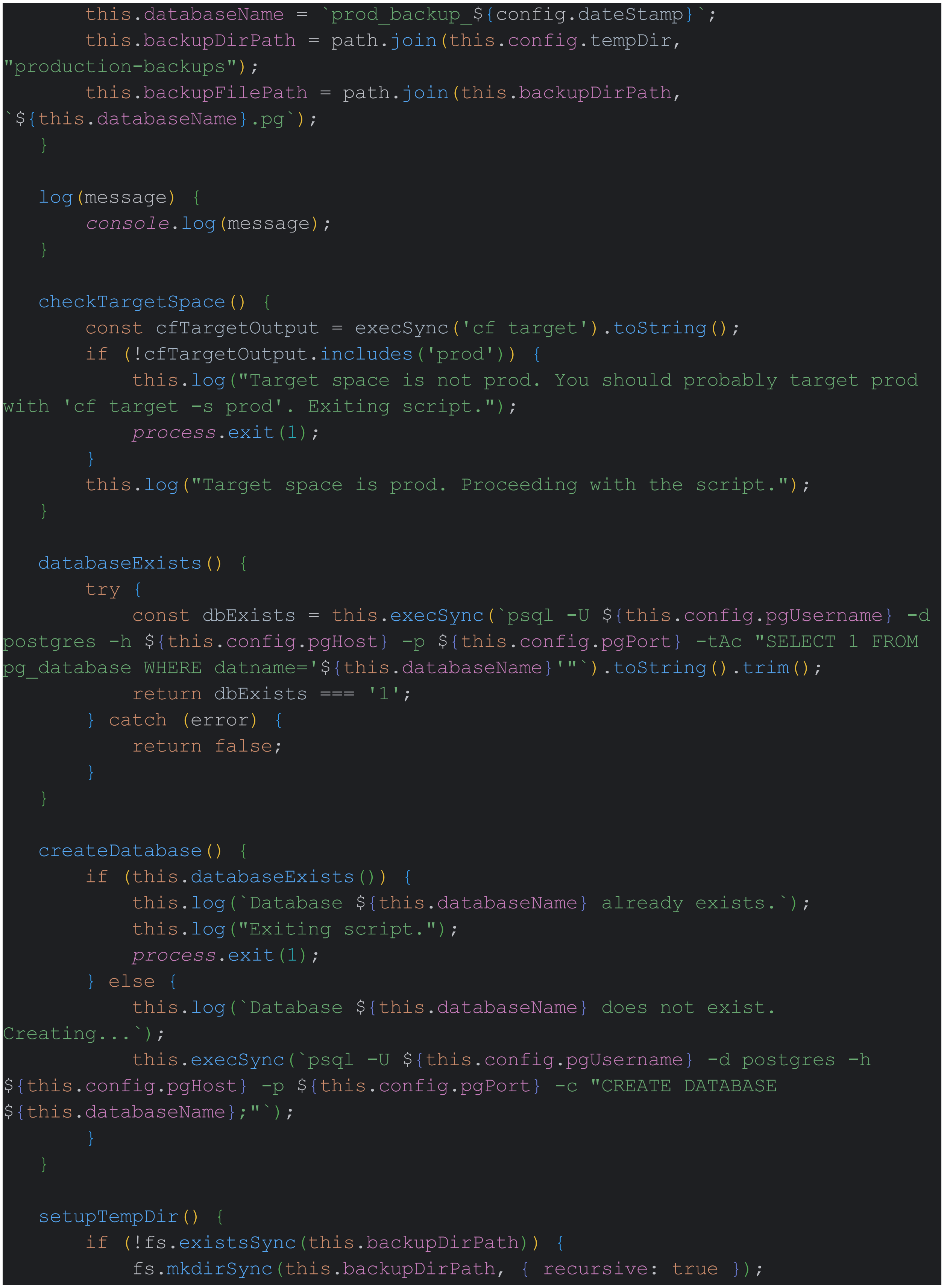
```
LOCAL_SNAPSHOT_FILE_NAME="weekly-snapshot-all-${CURRENT_DATE}.csv"

# Check if local file exists first
if [ -f "$SAVE_DIR/$LOCAL_SNAPSHOT_FILE_NAME" ]; then
   echo "Local snapshot file already exists:
$SAVE_DIR/$LOCAL_SNAPSHOT_FILE_NAME"
   echo "Exiting script."
   exit 1
fi

# Download the latest snapshot
echo "Downloading the latest snapshot from S3..."
aws s3api get-object --bucket ${S3_BUCKET_NAME} --key ${SNAPSHOT_FILE_NAME}
${SAVE_DIR}/${LOCAL_SNAPSHOT_FILE_NAME}

# Check if the download was successful
if [ -f "$SAVE_DIR/$LOCAL_SNAPSHOT_FILE_NAME" ]; then
   echo "Downloaded snapshot file: $SAVE_DIR/$LOCAL_SNAPSHOT_FILE_NAME"
else
   echo "Failed to download the snapshot file. Please check the error message
above."
   exit 1
fi
```

DatabaseSyncService.js

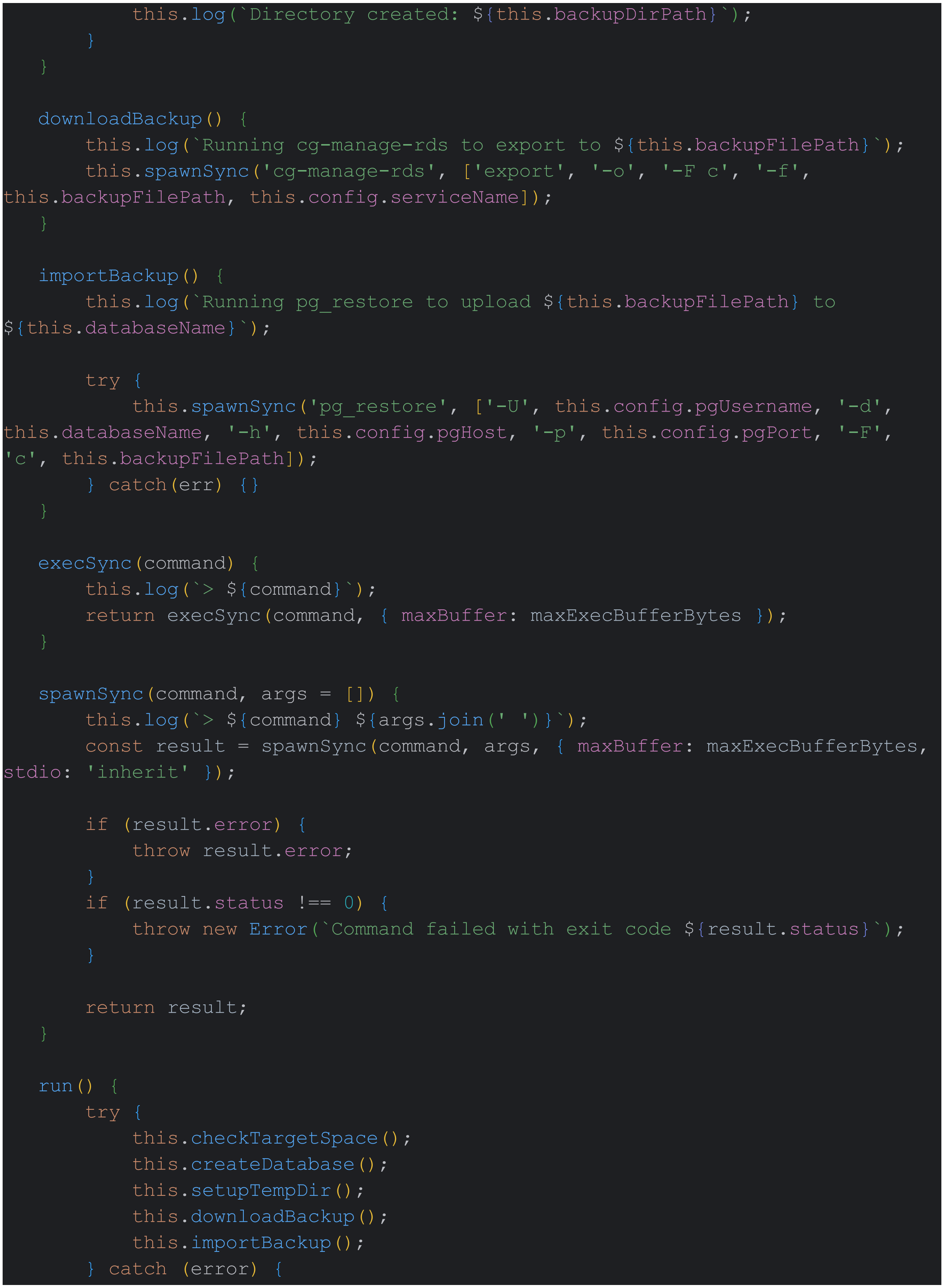
While not runnable itself (it must be invoked by other code), this JavaScript can be used to pull the production database and/or push the downloaded data into a locally-running Postgres server.

It would take a small amount of tweaking to get it working again, but hopefully having it may save someone some time.

```
const { execSync, spawnSync } = require('child_process');
const fs = require('fs');
const path = require('path');

const kilobyte = 1024;
const megabyte = 1024 * kilobyte;
const maxExecBufferBytes = 10 * megabyte; // 10 MB

class DatabaseSyncService {
   constructor(config) {
       this.config = config;
```



```
        this.databaseName = `prod_backup_${config.dateStamp}`;
        this.backupDirPath = path.join(this.config.tempDir,
"production-backups");
        this.backupFilePath = path.join(this.backupDirPath,
`${this.databaseName}.pg`);
    }

    log(message) {
        console.log(message);
    }

    checkTargetSpace() {
        const cfTargetOutput = execSync('cf target').toString();
        if (!cfTargetOutput.includes('prod')) {
            this.log("Target space is not prod. You should probably target prod
with 'cf target -s prod'. Exiting script.");
            process.exit(1);
        }
        this.log("Target space is prod. Proceeding with the script.");
    }

    databaseExists() {
        try {
            const dbExists = this.execSync(`psql -U ${this.config.pgUsername} -d
postgres -h ${this.config.pgHost} -p ${this.config.pgPort} -tAc "SELECT 1 FROM
pg_database WHERE datname='${this.databaseName}'"`).toString().trim();
            return dbExists === '1';
        } catch (error) {
            return false;
        }
    }

    createDatabase() {
        if (this.databaseExists()) {
            this.log(`Database ${this.databaseName} already exists.`);
            this.log("Exiting script.");
            process.exit(1);
        } else {
            this.log(`Database ${this.databaseName} does not exist.
Creating...`);
            this.execSync(`psql -U ${this.config.pgUsername} -d postgres -h
${this.config.pgHost} -p ${this.config.pgPort} -c "CREATE DATABASE
${this.databaseName};"`);
        }
    }

    setupTempDir() {
        if (!fs.existsSync(this.backupDirPath)) {
            fs.mkdirSync(this.backupDirPath, { recursive: true });
```



```
            this.log(`Directory created: ${this.backupDirPath}`);
        }
    }

    downloadBackup() {
        this.log(`Running cg-manage-rds to export to ${this.backupFilePath}`);
        this.spawnSync('cg-manage-rds', ['export', '-o', '-F c', '-f',
this.backupFilePath, this.config.serviceName]);
    }

    importBackup() {
        this.log(`Running pg_restore to upload ${this.backupFilePath} to
${this.databaseName}`);

        try {
            this.spawnSync('pg_restore', ['-U', this.config.pgUsername, '-d',
this.databaseName, '-h', this.config.pgHost, '-p', this.config.pgPort, '-F',
'c', this.backupFilePath]);
        } catch(err) {}
    }

    execSync(command) {
        this.log(`> ${command}`);
        return execSync(command, { maxBuffer: maxExecBufferBytes });
    }

    spawnSync(command, args = []) {
        this.log(`> ${command} ${args.join(' ')}`);
        const result = spawnSync(command, args, { maxBuffer: maxExecBufferBytes,
stdio: 'inherit' });

        if (result.error) {
            throw result.error;
        }
        if (result.status !== 0) {
            throw new Error(`Command failed with exit code ${result.status}`);
        }

        return result;
    }

    run() {
        try {
            this.checkTargetSpace();
            this.createDatabase();
            this.setupTempDir();
            this.downloadBackup();
            this.importBackup();
        } catch (error) {
            //this.log(`Error: ${error.message}`);
            this.log(error);
            process.exit(1);
        }
    }
}

module.exports = { DatabaseSyncService };
```

Luke Chavers wuz here XOX ;)
“Gone, but not forgotten”