Demonstration of Synthetic and Confidential Data Processing

We demonstrate a simple scenario of using containers (in this case, Docker) hosted on cloud platform (in this case, CodeOcean) to faciliate synthetic and confidential data processing. The purpose of using containers is to provide users with access to data and coding resources such that their analysis is easily reproducible, while retaining portability and adaptability on the data provider's side to facilitate validation. The use of cloud providers removes the need for users to install anything locally. CodeOcean is a commercial service facilitating that process by making the resources available through a web browser, though the basic functionality can be achieved on any container system. Options include Wholetale, Gigantum, and others. Finally, users who wish to not use such services can also typically provide their own setup, at very little additional cost or effort.

Reproducibility is important for synthetic data products when there is a validation or verification process involved. In such a setting, data users will first use the synthetic data to build and test their code for a desired analysis. Once the user is satisfied with the code used to perform their analysis, they can request validation or verification, and their code will be run by the data provider on the confidential data. Output from the analysis on the confidential data will then have to satisfy the data provider's disclosure avoidance procedures before it can be released to the user. In the case of validation, the results from the second run on confidential data are released to the user, possibly confidentialized. In the case of a verification server, the user only receives a (non-disclosive) message indicating whether her results from the first run can be considered to be inferentially valid or not.

One problem that can arise when maintaining a validation process is that the computing environment on the data user’s end does not exactly match the computing environment for the internal validation. This can lead to validation attempts that fail to run correctly or at all, which increases the wait time for data users and the staff time involved in performing the validation at the data provider. In order for this process to run smoothly, the data user's code needs to be reproducible. This ensures that the user’s code can be easily transferred and run on the confidential data. This can be achieved by using a "container." A container collects all of the necessary libraries, dependencies, and code needed to run an analysis in a single package. This container can then be downloaded to any machine and used with the local system to run the application. Crucially, container systems usually have the capability to either rebuild containers in a trusted environment, or to provide to users pre-configured secure containers that are also authorized to run in the secure area hosting the confidential data. In most cases, the core container itself does not need to be transferred, only a comprehensive recipe to build such a container.

In this document, we walk through the process for a specific use case. The Census Bureau, data owner of the confidential SIPP file merged with administrative data known as the SIPP Gold Standard file, makes a synthetic version of the Gold Standard file available as the "SIPP Synthetic Beta file" (SSB). The container host in the public domain is CodeOcean, which provides cloud-based access to Docker-based containers. Users log in via a simple web browser, with no install required. Crucially, CodeOcean provides access to Stata and Matlab compute capsules, covering 95% of economists' software needs.

In this configuration, the Census Bureau can use CodeOcean to house both the synthetic dataset as well as setup and configuration code that will assist the data user in creating code that is reproducible. The data user can then build on the setup and configuration code provided by the Census Bureau to perform their desired analysis. To perform the analysis on the synthetic data, users may either run the code directly on CodeOcean by paying for access to computing resources (CodeOcean also provides a limited amount of storage space and computing time for free), or they may download the compute capsule to their local machine and use their own computing resources. When the user requests validation, the Census Bureau can download the user’s compute capsule, securely rebuild the container (if necessary), and make minimal changes to excecute the analysis in a secure computing environment with the confidential data. By ensuring that everything can run within the CodeOcean compute capsule using the synthetic data, it also ensures that everything can be run within the associated compute capsule's container using the confidential data, even when the confidential data is not housed on CodeOcean or any publicly accessible service. In addition to providing a location for the Census Bureau and data users to share access to the synthetic dataset and code, it also ensures that the analysis will run correctly for both the data user and on the confidential data.

Development of code on open compute servers using synthetic data

This repository itself is the code. The code is written in Stata, and is confirmed to run on in this compute capsule.

Basic structure (refs) imposes a strict separation between code, immutable data, and reproducible results:

- all code is under `/code`
- all data is under `/data`
- all outputs MUST be written to `/results`, otherwise they are lost.

Outputs are regenerated at each run, and the history of such runs can be found (for the developing user) in the right pane. When the user, at the end of the development process, requests publication of the compute capsule, only the last run is published, together with code and data.

To facilitate this file organization for the user, the template code includes a config.do file, which applies best programming practices by defining some global variables for file paths that are used elsewhere in the code. It also instantiates logfiles, which are also written to `/results`.

The use of external packages

Stata, R, and many other programming languages use external packages of code to augment native capabilities. Initially, these need to be installed over the internet. However, there are various ways to address the issue at subsequent installations:

1. Packages can always be re-installed from source
2. Packages can be installed by a programming-language specific install script, and stored locally.
3. Packages can be incorporated into the container image itself.

CodeOcean has the ability to do the third method, via a "post-install" script and environment setup (see here and here). It can also support the first method during runtime (while connected to the internet). Because each run of Stata is transitory, there is no easy way to accomodate the second method.

We note that while hosted on CodeOcean, the same image, once built, is re-used, and packages are not re-installed. However, when exporting a capsule, only the build script is exported, and a replicator would need to rebuild the container, thereby also re-installing any packages. This can lead to version discrepancies when packages cannot be pinned to a particular version (as is the case with Stata).

Concrete example: Estimating economic returns to education in the SIPP

This example runs a Mincer equation on the SIPP Synthetic Beta data (need cite). The code is split into 4 pieces, tied together by a script. The environment is specified through a Dockerfile.

Script

The script run ties all Stata programs together. An alternative would be to have both a generic run, plus a master Stata do file. Both options work.

Stata programs

- config.do creates various global variables, and initializes a per-program log file, which will show up in `/results`.
- 00_setup.do initializes the code setup for each run.
    - It could also install Stata packages from SSC (see discussion above).
- 01_stats.do computes a few simple descriptive stats. Only log output is generated, but this could generate a summary stats table ("Table 1") of a paper.
- 02_mincer.do does data prep and runs a Mincer regression. Output is generated through `esttab`, and stored in `/results/mincer_results.csv`.

Dockerfile

The Dockerfile in this case specifies the use of a CodeOcean-specific pre-built Stata container, and handles installation of any Stata packages:

```
FROM registry.codeocean.com/codeocean/stata:16.0-ubuntu18.04

ARG DEBIAN_FRONTEND=noninteractive

COPY stata.lic /usr/local/stata/stata.lic
RUN stata 'ssc install estout' \
    && stata 'ssc install outreg' # Original versions: latest latest
```

Executing code on public (synthetic) data

Once the user has developed all the code, they execute a "reproducible run" on CodeOcean. This ensures that all code executes without error (note that it does *not* ensure that all necessary code has run - code can be commented out or be non-functional). This particular example, when run on CodeOcean infrastructure in 2021, takes about 4 minutes to execute.

Alternatively, the user can export the entire capsule (including data), rebuild the image locally, and execute on their local infrastructure, using an unmodified

Building the container:

```
cd /path/to/downloaded/capsule/environment
VERSION=16
TAG=$(date +%F)
MYHUBID=larsvilhuber
MYIMG=ssb-demo
DOCKER_BUILDKIT=1 docker build  . -t $MYHUBID/${MYIMG}:$TAG
[+] Building 5.9s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 365B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for registry.codeocean.com/codeocean/stata:1
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 133B
0.0s
 => [1/3] FROM registry.codeocean.com/codeocean/stata:16.0-ubuntu18.04
0.0s
 => CACHED [2/3] COPY stata.lic /usr/local/stata/stata.lic
0.0s
 => [3/3] RUN stata 'ssc install estout'     && stata 'ssc install outreg
5.8s
 => exporting to image
0.0s
 => => exporting layers
0.0s
 => => writing image sha256:1701246f8ee5582afd6b4606d9f5cf1c5bb43c1eb4e06
0.0s
 => => naming to docker.io/larsvilhuber/ssb-demo:2021-10-06
0.0s
```

Running the container:

```
cd /path/to/downloaded/capsule/
docker run -it --rm \
  -v $(pwd)/code:/code \
  -v $(pwd)/data:/data \
  -v $(pwd)/results:/results \
  -w /code \
  $MYHUBID/${MYIMG} ./run
```

which runs for about 3 minutes on a 2021-vintage Linux workstation.

Porting to confidential compute server

In order to conduct a validation exercise, the code needs to be re-executed in the secure data environment. The compute capsule is exported (via "Capsule -> Export"), which provides a full package. Since exporting the package is done here by the data owner, exporting the data is not necessary, making for a light package. Alternatively, the code can also be downloaded via `git clone`.

Modifying code

As configured in the present example, the code requires only minor modifications to work on confidential data. The provided `config.do` contains switches that handle the setup:

```
global confidential no

/* SSB parameters */
if ( "$confidential" == "no" ) {
    global SSBtype synthetic
    global inputdata "../data"
}
if ( "$confidential" == "yes" ) {
    global SSBtype confidential
    global inputdata "/confidential/data" // This needs to be mounted when
running the capsule!
    // other confidential parameters are stored outside of this file
    include "config-confidential.do"
}
```

This can be easily algorithmically modified, e.g., `sed -i 's/confidential no/confidential yes/' config.do`.

Alternate methods exist as well. For instance, one could test for presence of "`config-confidential.do`" and include it if present, overriding any parameters in the main `config.do`.

Modifying the container base image

The CodeOcean capsule uses a CodeOcean-specific prebuilt container used to execute the code (in the case of this capsule, `registry.codeocean.com/codeocean/stata:16.0-ubuntu18.04`). This image will need to be replaced with a container that satisfies the data owner's security requirements, while maintaining full compatibility with the needs of the environment. Because this example uses Stata, which behaves fairly uniformly across various Linux installs, the particular version of the Linux base image is likely not important. Alternatively, the validation exercise can be coordinated with the provider, and the provider can offer a generic security-vetted image that is verified to be functionally equivalent to the image used in the secure environment. Finally, the Docker file underlying the `stata:16.0-ubuntu18.04` image, which builds the container from scratch, can be used to rebuild a container within the secure environment (see github.com/AEADataEditor/docker-stata/releases/tag/stata16-2021-06-09 for an example). At scale, this would simply use a similar, security-vetted, pre-built container, e.g., `registry.census.gov/codeocean/stata:16.0-ubuntu18.04-secure`.

The key feature here is that no binary code needs to be transferred into the secure environment, eliminating a security risk. The execution environment is completely known to the IT personnel of the data provider. Only the user-provided Stata code is needed for the validation. Since execution is in a controlled environment, and can be trivially separated from othe sensitive areas (code cannot "break out" of the container), security is substantially enhanced.

Modifying the input data

Finally, the synthetic input data available in the public-facing environment needs to be replaced by confidential data. In the Stata code, this is already handled, as outlined above. In order to make this actionable, the Docker image can be executed in a particular fashion, provisioning the container with confidential data.

Executing code on confidential data

A (simulated) example for a hypothetical request recorded as "Request 12345" is shown below:

```
VERSION=16
TAG=16.0-ubuntu18.04-secure
MYHUBID=censusbureau
MYIMG=stata
STATALIC=/path/to/stata/licenses
CONFDATA=/path/to/confidential/data
REQUEST=12345
cd /path/to/requests
docker run -it --rm \
  -v ${STATALIC}/stata.lic.${VERSION}:/usr/local/stata/stata.lic \
  -v $(pwd)/$REQUEST/code:/code \
  -v $(pwd)/$REQUEST/data:/data \
  -v $(pwd)/$REQUEST/results-confidential:/results \
  -v $CONFDATA/data:/confidential/data \
  -v $CONFDATA/config-confidential.do:/code/config-confidential.do \
  $MYHUBID/${MYIMG} run
```

The example replicates the directory structure from the public CodeOcean capsule, but also bind-mounts into the container the confidential data (`$CONFDATA/data`) and the configuration file for the confidential data (`config-confidential.do`). The `$CONFDATA` area therefore might look like this:

```
/path/to/confidential/data:
  - ssb_v7_0_confidential1.dta
  - ssb_v7_0_confidential2.dta
  - ssb_v7_0_confidential3.dta
  - ssb_v7_0_confidential4.dta
  - ssb_v7_1_confidential1.dta
  ...
  - config-confidential.do
```

Requests are stored in a particular area, separately for each request. Results are written into a "`results-confidential`" directory, denoting that they have not yet been vetted by data provider's disclosure avoidance procedures.

Of note is that the above functionality can be easily automated. No manual intervention is necessary. Scripts can check for execution errors and report these back to the authors or to staff. Successful completion triggers the next part of the workflow.

Sending results back to user

Once results have been generated, the usual disclosure avoidance workflow at the data provider is triggered. This might entail post-processing of the results, generation of additional supporting statistics (though these should generally be included in the processing), and finally, provision of the results to users. Whether that part can be automated or rationalized is not unique to the validation process.

Conclusion

The use of containers ensures reproducibility, as well as reliable portability. The use of cloud-based commercial services requires no infrastructure or software maintenance by either data provider or users. With very little effort, automation is possible (potentially through web forms), and the only likely constraint to full automation is the absence of automated output vetting algorithms.