Tapis CLI How-to Guide

Release 1.0.0.0a10

May 26, 2023

Getting Started:

The Tapis CLI is python-based tooling for interacting with the Tapis Platform. Below is a series of short guides to accomplishing common tasks with the Tapis CLI. Most guides assume you already have the CLI installed and that you are authenticated with one of the tenants. Additional help is located in the Tapis CLI Reference Documentation.

CHAPTER 1

Install the CLI

The Tapis CLI is available as a Python package. We highly recommend using Python 3.7+ as the Python runtime behind the Tapis CLI. We support Python 2.7 for legacy applications, but on a best-effort basis as Python 2.7 is a deprecated language.

1.1 Install with Pip

```
$ pip install tapis-cli
```

1.2 Install from Source

```
$ git clone https://github.com/TACC-Cloud/tapis-cli-ng.git
$ cd tapis-cli-ng/
$ pip install --upgrade --user .
```

1.3 Run in a Container

```
$ docker run --rm -it -v ${PWD}:/work -v ${HOME}/.agave:/root/.agave \
      tacc/tapis-cli-ng:latest /bin/bash
```

CHAPTER 2

Request Access to a Tenant

Each tenant has its own criteria for admitting access. By default, anyone can get access to the **tacc.prod** tenant by creating a TACC Account.

Please visit the following pages to apply for access to other tenants as appropriate:

Name	URL
3DEM	https://3dem.org/
Agave Public Tenant	https://public.agaveapi.co/
Araport	https://www.araport.org/
CyVerse Science APIs	https://cyverse.org/Science-APIs
DesignSafe	https://www.designsafe-ci.org/
Science Gateways Community Institute	https://sciencegateways.org/
SD2E	https://sd2e.org/
TACC Prod	https://portal.tacc.utexas.edu
UTRC Portal	https://utrc.tacc.utexas.edu/
VDJ Server	https://vdjserver.org/

CHAPTER 3

Initialize a Session

You must set up a Tapis session on each host where you will use the Tapis CLI. This is a scripted process implemented by the command `tapis auth init`. This is a one-time operation where you will be asked to agree to terms, select a tenant, and finally enter a username and password for that tenant.

```
$ tapis auth init

Use of Tapis requires acceptance of the TACC Acceptable Use Policy
which can be found at https://portal.tacc.utexas.edu/tacc-usage-policy

Do you agree to abide by this AUP? (type 'y' or 'n' then Return) y

Use of Tapis requires acceptance of the Tapis Project Code of Conduct
which can be found at https://tapis-project.org/code-conduct

Do you agree to abide by this CoC? (type 'y' or 'n' then Return) y

To improve our ability to support Tapis and the Tapis CLI, we would like to
collect your IP address, operating system and Python version. No personally-
identifiable information will be collected. This data will only be shared in
aggregate form with funders and Tapis platform stakeholders.

Do you consent to this reporting? [Y/n]: Y
+----------------+---------------------------------------+------------------------------
↪----------+
|      Name      |              Description              |                  URL           ␣
↪          |
+----------------+---------------------------------------+------------------------------
↪----------+
|      3dem      |              3dem Tenant              |         https://api.3dem.org/
↪          |
|   agave.prod   |          Agave Public Tenant          |      https://public.agaveapi.
↪co/        |
|  araport.org   |                Araport                |        https://api.araport.
↪org/         |
```

(continues on next page)

(continued from previous page)

```
|     bridge    |                 Bridge                 |     https://api.bridge.tacc.
↪cloud/     |
|   designsafe  |               DesignSafe               |    https://agave.designsafe-
↪ci.org/    |
|  iplantc.org  |          CyVerse Science APIs          |       https://agave.iplantc.
↪org/       |
|      irec     |               iReceptor                | https://irec.tenants.prod.
↪tacc.cloud/  |
|    portals    |             Portals Tenant             |  https://portals-api.tacc.
↪utexas.edu/  |
|      sd2e     |              SD2E Tenant               |         https://api.sd2e.org/
↪          |
|      sgci     | Science Gateways Community Institute |        https://sgci.tacc.
↪cloud/        |
|   tacc.prod   |                  TACC                  |      https://api.tacc.utexas.
↪edu/       |
| vdjserver.org |               VDJ Server               | https://vdj-agave-api.tacc.
↪utexas.edu/ |
+---------------+--------------------------------------+------------------------------
↪----------+
Enter a tenant name [tacc.prod]:
tacc.prod username: taccuser
tacc.prod password for taccuser:
```

Session tokens and other metadata are stored in `~/.agave/config.json` as well as in `~/.env`.

CHAPTER 4

Interact with Systems

A Tapis **system** is a server or collection of servers associated with a single hostname. They may be public or private, and they may either be **storage** systems (used for storing files) or **execution** systems (used for running jobs).

On some tenants, users are provided private storage and execution systems attached to TACC resources. If you would like to create your own systems, skip ahead to Create a Private System.

Warning: The **tacc.prod** tenant likely does not contain default systems for your username. Skip ahead to Create a Private System before working through this guide.

4.1 Find Systems

Systems can be discovered using the `tapis systems list` command:

On this tenant, the user `taccuser` sees numerous storage and execution systems. All systems are either public or private (with varying degrees of privacy), each of which are shown here.

On tenants with many more systems, it may be useful to narrow the list with the `tapis systems search` command. For example, to discover systems only owned by you:

```
$ tapis systems search --owner eq taccuser
+--------------------------+----------------------------------------+-----------+------
↪---+
| id                       | name                                   | type      |␣
↪default |
+--------------------------+----------------------------------------+-----------+------
↪---+
| tacc.stampede2.taccuser | Execution system for TACC Stampede2    | EXECUTION |␣
↪False   |
| tacc.work.taccuser       | Storage system for TACC work directory | STORAGE   |␣
↪False   |
+--------------------------+----------------------------------------+-----------+------
↪---+
```

(continues on next page)

(continued from previous page)

Other useful searches might include:

```
$ tapis systems search --owner neq taccuser    # systems you don't own
$ tapis systems search --public eq true          # only public systems
$ tapis systems search --public eq false         # only private systems
$ tapis systems search --type eq EXECUTION       # only execution systems
$ tapis systems search --type eq STORAGE         # only storage systems
$ tapis systems search --default eq true         # your default system
```

You can also chain multiple search parameters. For example, to display only private storage systems:

```
$ tapis systems search --type eq STORAGE --public eq false
+--------------------+----------------------------------------+---------+---------+
| id                 | name                                   | type    | default |
+--------------------+----------------------------------------+---------+---------+
| tacc.work.taccuser | Storage system for TACC work directory | STORAGE | False   |
+--------------------+----------------------------------------+---------+---------+
```

Note: Don't forget to replace instances of **taccuser** with your actual username on the tenant

4.2 Discover System Hostname and Default Paths

To see additional system information, including hostname, root folder locations, and availability, use the `tapis systems show` command with a system ID. For example, find out more information about your personal storage system as follows:

```
$ tapis systems show -f json tacc.work.taccuser
```

```
{
  "id": "tacc.work.taccuser",
  "name": "Storage system for the TACC WORK directory",
  "type": "STORAGE",
  "default": false,
  "available": true,
  "description": "Storage system for the TACC WORK directory via Stampede2",
  "environment": null,
  "executionType": null,
  "globalDefault": false,
  "lastModified": "7 hours ago",
  "login": null,
  "maxSystemJobs": null,
  "maxSystemJobsPerUser": null,
  "owner": "taccuser",
  "public": false,
  "queues": null,
  "revision": 1,
  "scheduler": null,
  "scratchDir": null,
  "site": null,
  "status": "UP",
```

(continues on next page)

(continued from previous page)

```
  "storage": {
    "proxy": null,
    "protocol": "SFTP",
    "mirror": false,
    "port": 22,
    "auth": {
      "type": "SSHKEYS"
    },
    "publicAppsDir": null,
    "host": "stampede2.tacc.utexas.edu",
    "rootDir": "/work/01234/taccuser",
    "homeDir": "/"
  },
  "uuid": "3834240380791075626-242ac112-0001-006",
  "workDir": null
}
```

The `-f json` flag was provided to render all information about the system. As described above, this storage system is a gateway to your private storage space on the TACC *$WORK* filesystem. The *rootDir* is the virtual root path for operations performed on this system. The highlighted lines emphasize the host and root directory when performing operations against this system.

In addition to standard host and path information, execution systems also contain information about queue types and availability.

4.3 Check System Status

It may be useful to check the status (availability) of a system in a scriptable way prior to, e.g., uploading files as part of a pipeline. The following command can be used with extra flags to strip out the useful part of the response:

```
$ tapis systems show -c status -f value tacc.work.taccuser
UP
```

Note: Use `tapis command subcommand --help` to find usage information for each command

CHAPTER 5

Perform Basic File Operations

Data in the Tapis ecosystem can be managed using the `files` service. With many parallels to Unix-style commands (`ls, mv, cp, rm`, etc.), the `tapis files -` commands can be used to list files, upload and download data, remotely manage and organize data, and import external data from the web.

5.1 List Files and Navigate the File Tree

To list the files available to you on a storage system, use:

```
$ tapis files list agave://tacc.work.taccuser/
+-----------+--------------+--------+
| name      | lastModified | length |
+-----------+--------------+--------+
| jobs      | 2 years ago  |   4096 |
| maverick  | 2 years ago  |   4096 |
| stampede2 | 2 years ago  |   4096 |
| wrangler  | 2 years ago  |   4096 |
+-----------+--------------+--------+
```

The URI provided on the command line (`agave://tacc.work.taccuser/`) takes the form:

1. `agave://` => refer to an Agave URI
2. `tacc.work.taccuser` => the name of the storage system
3. `/` => the relative path from the root directory on the storage system

To make a new folder, then list the contents of that folder:

```
$ tapis files mkdir agave://tacc.work.taccuser/ test_folder
+--------------+---------------------------------------+
| Field        | Value                                 |
+--------------+---------------------------------------+
| name         | test_folder                           |
```

(continues on next page)

(continued from previous page)

```
| uuid         | 2668156827089366550-242ac112-0001-002 |
| owner        | taccuser                              |
| path         | /test_folder                          |
| lastModified | 2020-05-12T07:48:19.141-05:00         |
| source       | None                                  |
| status       | STAGING_COMPLETED                     |
| nativeFormat | dir                                   |
| systemId     | tacc.work.taccuser                    |
+--------------+---------------------------------------+

$ tapis files list agave://tacc.work.taccuser/test_folder/
      # currently empty
```

To remove a folder, use the `tapis files delete`:

```
$ tapis files delete agave://tacc.work.taccuser/test_folder
+--------------+-------+
| Field        | Value |
+--------------+-------+
| deleted      | 1     |
| skipped      | 0     |
| warnings     | 0     |
| elapsed_msec | 2318  |
+--------------+-------+
```

Warning: The `tapis files delete` command will delete folders with or without contents.

5.2 Upload and Download Files

Files can be transferred from your local machine to the remote storage system using `tapis files upload`, and from the remote storage system to your local machine using the `tapis files download`.

First, find or create a local file and upload it to the storage system (recreate the `test_folder/` if you deleted it in the previous example):

```
$ touch local_file.txt
$ echo 'Hello, world!' > local_file.txt
$ tapis files upload agave://tacc.work.taccuser/test_folder local_file.txt
+-------------------+-------------+
| Field             | Value       |
+-------------------+-------------+
| uploaded          | 1           |
| skipped           | 0           |
| messages          | 0           |
| bytes_transferred | 14.00 bytes |
| elapsed_sec       | 2           |
+-------------------+-------------+

$ tapis files list agave://tacc.work.taccuser/test_folder/
+-----------------+----------------+--------+
| name            | lastModified   | length |
+-----------------+----------------+--------+
```

(continues on next page)

(continued from previous page)

```
| local_file.txt | 26 seconds ago |     14 |
+----------------+----------------+--------+
```

Use `tapis files copy` to make a copy of the file on the remote system:

```
$ tapis files copy agave://tacc.work.taccuser/test_folder/local_file.txt /test_folder/
↪remote_copy.txt
+--------------+--------------------------------------------------------------------
↪----------------------------+
| Field        | Value                                                              ␣
↪                            |
+--------------+--------------------------------------------------------------------
↪----------------------------+
| name         | remote_copy.txt                                                    ␣
↪                            |
| uuid         | 6484805032038306282-242ac112-0001-002                              ␣
↪                            |
| owner        | taccuser                                                           ␣
↪                            |
| path         | /test_folder/remote_copy.txt                                       ␣
↪                            |
| lastModified | 2020-05-12T07:51:52.187-05:00                                      ␣
↪                            |
| source       | https://api.tacc.utexas.edu/files/v2/media/system/tacc.work.taccuser/
↪/test_folder/local_file.txt |
| status       | STAGING_COMPLETED                                                  ␣
↪                            |
| nativeFormat | raw                                                                ␣
↪                            |
| systemId     | tacc.work.taccuser                                                 ␣
↪                            |
+--------------+--------------------------------------------------------------------
↪----------------------------+

$ tapis files list agave://tacc.work.taccuser/test_folder
+-----------------+---------------+--------+
| name            | lastModified  | length |
+-----------------+---------------+--------+
| local_file.txt  | 7 minutes ago |     14 |
| remote_copy.txt | 3 minutes ago |     14 |
+-----------------+---------------+--------+
```

Note that the second argument provided on the command line contains both the name of the copied file, and the full path relative to the root directory for the storage system.

To download the result:

```
$ tapis files download agave://tacc.work.taccuser/test_folder/remote_copy.txt
$ ls
local_file.txt    remote_copy.txt
$ cat remote_copy.txt
Hello, world!
```

Note: Use the `-W` flag to recursively download the contents of a whole directory

5.3 Other File Operations

Using the Tapis CLI, files and folders can also be renamed, moved, and deleted remotely on the storage system. The syntax for these operations is very similar to the `tapis files copy` command syntax. Here are some common examples:

```
# Rename a file in place
$ tapis files move agave://tacc.work.taccuser/test_folder/remote_copy.txt /test_
↪folder/renamed.txt

# Make a subfolder in the test_folder/ folder
$ tapis files mkdir agave://tacc.work.taccuser/test_folder/ subfolder

# Rename a folder in place
$ tapis files move agave://tacc.work.taccuser/test_folder/subfolder /test_folder/
↪renamed_folder

# Move a file into that subfolder
$ tapis files move agave://tacc.work.taccuser/test_folder/renamed.txt /test_folder/
↪renamed_folder/renamed.txt

# Delete a file or a folder
$ tapis files delete agave://tacc.work.taccuser/test_folder/renamed_folder
```

Be cautious with `tapis files move` and `tapis files delete` commands. Just like a Linux filesystem, files inadvertently deleted or overwritten are most likely unrecoverable.

5.4 File or Folder History

You can list the history of events for a specific file or folder. This will give more descriptive information (when applicable) related to number of retries, permission grants and revocations, reasons for failure, and hiccups that may have occurred in the transfer process.

```
$ tapis files history agave://tacc.work.taccuser/test_folder/local_file.txt
+--------------------+---------------+-----------------------------------------------
↪------------------------------+
| status             | created       | description                                  ␣
↪                              |
+--------------------+---------------+-----------------------------------------------
↪------------------------------+
| STAGING_QUEUED     | 6 minutes ago | File/folder queued for staging               ␣
↪                              |
| STAGING_COMPLETED  | 6 minutes ago | Your scheduled transfer of http://129.114.97.
↪130/local_file.txt completed     |
|                    |               | staging. You can access the raw file on Storage␣
↪system for TACC work          |
|                    |               | directory at /work/01234/taccuser/test_folder/
↪local_file.txt or via the API    |
|                    |               | at https://api.tacc.utexas.edu/files/v2/media/
↪system/tacc.work.taccuser//test |
|                    |               | _folder/local_file.txt.                      ␣
↪                              |
| DOWNLOAD           | 4 minutes ago | File was downloaded                          ␣
↪                              |
+--------------------+---------------+-----------------------------------------------
↪------------------------------+
```

(continues on next page)

(continued from previous page)

5.5 Further Help

Reminder: at any time, you can issue a Tapis CLI command with the `-h` flag to find more information on the function and usage of the command. Extensive Tapis CLI documentation can be found HERE.

CHAPTER 6

Import Data from Alternative Sources

Warning: This functionality does not yet exist

The Tapis platform enables management of multiple storage systems representing different hosts under the same tenant namespace. Data can be moved efficiently directly between hosts (storage systems).

6.1 Import Files from other Systems

To import data from other storage systems, e.g. from the community data space to your private data space, use `tapis files import`:

```
$ tapis files import agave://community-data/public_file.txt  /
                     agave://private-system/destination_folder/
```

With the above syntax, the file located at the root directory on the `community-data` storage system will be imported to your private storage system, and placed in your directory `destination_folder`.

Please also note that even though you are *able* to import files from other Tapis storage systems, you may not always *need* to import those files. Most applications of Tapis will allow you to provide the complete URI path to the file, e.g. `agave://community-data/public_file.txt`. This is useful, for example, in the case of large reference libraries. Pointing to the remote libraries rather than copying them saves time and disk space.

6.2 Import Files from the Web

You can also import files from the web using the URL. This is useful to import files that are not part of an existing Tapis storage system:

```
$ tapis files import https://website.com/raw_file \
                     agave://private-system/destination_folder
```

CHAPTER 7

Share Data with Others

The Tapis CLI makes it possible to share data with other users on the same tenant. Use your best judgement in deciding whether to copy shared data or link against shared data with the understanding that storage space is limited. The guide below demonstrates how to modify permissions on a given data set to share it in place without copying.

7.1 Find Another User

To share files with another user on the same tenant, you must first know their username. The Tapis CLI has a set of tools that can be used to find other users. View your own user profile by issuing:

```
$ tapis profiles show self
+--------------+---------------------+
| Field        | Value               |
+--------------+---------------------+
| first_name   | Tacc                |
| last_name    | User                |
| email        | taccuser@gmail.com  |
| mobile_phone |                     |
| phone        |                     |
| username     | taccuser            |
+--------------+---------------------+
```

Each of the fields stored in the user profile is queryable using the `tapis profiles search` command. Some more common examples include:

```
# Search for another user by first name
$ tapis profiles search --first-name eq John

# Search for another user by last name
$ tapis profiles search --last-name eq Doe

# Search for another user by email address
$ tapis profiles search --email eq jdoe@utexas.edu
```

Once you have identified the correct username, you can query it to make sure it is the person you are looking for:

```
$ tapis profiles show jdoe
+--------------+-----------------+
| Field        | Value           |
+--------------+-----------------+
| first_name   | John            |
| last_name    | Doe             |
| email        | jdoe@utexas.edu |
| mobile_phone |                 |
| phone        |                 |
| username     | jdoe            |
+--------------+-----------------+
```

7.2 Share Files with Another User

File permissions are managed similar to Unix file permissions. To list the permissions on an existing file on one of your storage systems, issue:

To add permissions for another user (with username `jdoe`) to read the file, use the `tapis files pems grant` with the following positional arguments:

Warning: Recursive permission changes are not yet implemented

Now, a user with username `jdoe` has permissions to read the file. Valid values for setting permission are ALL, READ, WRITE, READ_WRITE, EXECUTE, READ_EXECUTE, WRITE_EXECUTE, and NONE. However, before `jdoe` can access the file, they also need permissions on the private storage system. To see who has access to your storage system, perform:

```
$ tapis systems roles list tacc.work.taccuser
+----------+-------+
| username | role  |
+----------+-------+
| taccuser | OWNER |
+----------+-------+
```

To add your collaborator to your system use:

```
$ tapis systems roles grant tacc.work.taccuser jdoe GUEST
+----------+---------+
| Field    | Value   |
+----------+---------+
| username | jdoe    |
| role     | GUEST   |
+----------+---------+

$ tapis systems roles list tacc.work.taccuser
+----------+-------+
| username | role  |
+----------+-------+
| taccuser | OWNER |
| jdoe     | GUEST |
+----------+-------+
```

Now, a user with username `jdoe` can see files with the appropriate permissions on your storage system. Valid values for setting a role include GUEST, USER, PUBLISHER, ADMIN, and OWNER.

Finally, ask your collaborator to download the file with the exact same command you use to download the file:

```
$ tapis files download agave://tacc.work.taccuser/test_folder/local_file.txt
```

7.3 Revoke Permissions

If you want to revoke permissions, make sure to revoke permissions both on the shared file as well as the storage system:

```
# Revoke permissions on the shared file
$ tapis files pems revoke agave://tacc.work.taccuser/test_folder/local_file.txt jdoe

# Revoke permissions on the private storage system
$ tapis systems roles revoke tacc.work.taccuser jdoe
```

You can also blanket revoke permissions from all non-owner users:

```
# Revoke permissions on the shared file for all users
$ tapis files pems drop agave://tacc.work.taccuser/test_folder/local_file.txt

# Revoke permissions on the private storage system for all users
$ tapis systems roles drop tacc.work.taccuser
```

7.4 Share Files Using Postits

Another convenient way to share data is the Tapis postits service. Postits generate a short URL with a user-specified lifetime and limited number of uses. Anyone with the URL can paste it into a web browser, or curl against it on the command line. To create a postit:

```
$ tapis postits create -L 3600 -m 5 agave://tacc.work.taccuser/test_folder/file.txt
+---------------+---------------------------------------------------------------------
↪----------------------------+
| Field         | Value                                                               ␣
↪                            |
+---------------+---------------------------------------------------------------------
↪----------------------------+
| postit        | a88eed9c3bb7ae9f8dca6a8c1cc8c25f                                    ␣
↪                            |
| remainingUses | 5                                                                   ␣
↪                            |
| expires       | 2020-05-12T09:21:32-05:00                                           ␣
↪                            |
| url           | https://api.tacc.utexas.edu/files/v2/media/system/tacc.work.
↪taccuser/test_folder/local_file.txt |
| creator       | taccuser                                                            ␣
↪                            |
| created       | 2020-05-12T08:21:32-05:00                                           ␣
↪                            |
| noauth        | False                                                               ␣
↪                            |
```

(continues on next page)

(continued from previous page)

```
| method        | GET                                                                      ␣
↪                          |
| postit_url    | https://api.tacc.utexas.edu//postits/v2/
↪a88eed9c3bb7ae9f8dca6a8c1cc8c25f                           |
+---------------+------------------------------------------------------------------------
↪----------------------------+
```

The response from this command includes a URL which can be pasted into a web browser or curled on the command line:

```
https://api.tacc.utexas.edu//postits/v2/a88eed9c3bb7ae9f8dca6a8c1cc8c25f
```

This postit will work for 5 downloads (`-m 5`) and only available for one hour (3600 seconds, `-L 3600`). The creator of the postit can list and delete their postits with the following commands:

```
$ tapis postits list
+----------------------------------+---------------+----------------------------+------
↪-------------------------------------------------------------------------------------
↪-------+
| postit                           | remainingUses | expires                    | url ␣
↪                                                                                     ␣
↪       |
+----------------------------------+---------------+----------------------------+------
↪-------------------------------------------------------------------------------------
↪-------+
| a88eed9c3bb7ae9f8dca6a8c1cc8c25f |             4 | 2020-05-12T09:21:32-05:00 |␣
↪https://api.tacc.utexas.edu/files/v2/media/system/tacc.work.taccuser/test_folder/
↪local_file.txt |
+----------------------------------+---------------+----------------------------+------
↪-------------------------------------------------------------------------------------
↪-------+

$ tapis postits delete a88eed9c3bb7ae9f8dca6a8c1cc8c25f
```

CHAPTER 8

Find an Application

Each Tapis tenant has its own set of public applications ("apps" for short) that are available for tenant-authenticated users to run. Public apps are also tied to public execution systems.

8.1 List Available Applications

The `tapis apps list` command can be used to list all apps available to you. The **tacc.prod** tenant has the following apps available:

```
$ tapis apps list
+-----------------------------------+-------------------+
| id                                | label             |
+-----------------------------------+-------------------+
| tapis.app.imageclassify-1.0u3     | Image Classifier  |
| tapis.app.imageclassify-1.0u2     | Image Classifier  |
| tapis.app.imageclassify-1.0u1     | Image Classifier  |
| vina-ls5-1.1.2u3                  | Autodock Vina     |
| vina-ls5-1.1.2u2                  | Autodock Vina     |
| vina-ls5-1.1.2u1                  | Autodock Vina     |
| opensees-2.4.4-slurm-2.4.4.5804u1 | OpenSees          |
| opensees-fork-2.4.4.5804u2        | OpenSees          |
| opensees-2.4.4.5804u1             | OpenSees          |
| opensees-fork-2.4.4.5804u1        | OpenSees          |
| vina-lonestar-1.1.2u4             | Autodock Vina     |
| vina-lonestar-1.1.2u3             | Autodock Vina     |
| vina-lonestar-1.1.2u2             | Autodock Vina     |
| pdb2pdbqt-1.00u1                  | pdb2pdbqt         |
+-----------------------------------+-------------------+
```

Public apps will have a revision tag at the end (`u1`, `u2`, `u3` etc.). The higher the number, the newer the revision.

8.2 Search for Applications by Name

Use `tapis apps search` to search for apps on a number of different criteria. Some common searches might include:

```
# Search for an app by part of its name
$ tapis apps search --name like imageclassify

# Search for apps that you own
$ tapis apps search --owner eq taccuser

# Search for apps that you don't own (public or shared with you)
$ tapis apps search --owner neq taccuser
```

8.3 Display Application Information

Many applications you will find in the catalog can be used in multiple ways. It is up to the developer of an app to decide which function(s) of the particular tool the app will perform, as well as what are the expected inputs, parameters, and outputs. To see the description of an app use:

```
$ tapis apps show tapis.app.imageclassify-1.0u3
+---------------------------+-----------------------------------------------------------
↪--------+
| Field                     | Value                                                    ␣
↪        |
+---------------------------+-----------------------------------------------------------
↪--------+
| id                        | tapis.app.imageclassify-1.0u3                            ␣
↪        |
| name                      | tapis.app.imageclassify                                  ␣
↪        |
| version                   | 1.0                                                      ␣
↪        |
| revision                  | 3                                                        ␣
↪        |
| label                     | Image Classifier                                         ␣
↪        |
| lastModified              | 6 months ago                                             ␣
↪        |
| shortDescription          | Classify an image using a small ImageNet model           ␣
↪        |
| longDescription           |                                                          ␣
↪        |
| owner                     | cicsvc                                                   ␣
↪        |
| isPublic                  | True                                                     ␣
↪        |
| executionType             | CLI                                                      ␣
↪        |
| executionSystem           | tapis.execution.system                                   ␣
↪        |
| deploymentSystem          | docking.storage                                          ␣
↪        |
| available                 | True                                                     ␣
↪        |
```

(continues on next page)

(continued from previous page)

```
| parallelism             | SERIAL                                                        ␣
↪         |
| defaultProcessorsPerNode | 1                                                            ␣
↪         |
| defaultMemoryPerNode    | 1                                                             ␣
↪         |
| defaultNodeCount        | 1                                                             ␣
↪         |
| defaultMaxRunTime       | None                                                          ␣
↪         |
| defaultQueue            | None                                                          ␣
↪         |
| helpURI                 |                                                               ␣
↪         |
| deploymentPath          | /home/docking/api/v2/prod/apps/tapis.app.imageclassify-1.
↪0u3.zip |
| templatePath            | wrapper.sh                                                    ␣
↪         |
| testPath                | test/test.sh                                                  ␣
↪         |
| checkpointable          | False                                                         ␣
↪         |
| uuid                    | 3162334876895875561-242ac119-0001-005                         ␣
↪         |
| icon                    | None                                                          ␣
↪         |
+-------------------------+-----------------------------------------------------------
↪--------+
```

The output of this command is a table-formatted description of the app including select metadata. To see all of the app details including inputs, parameters, and outputs, use the `-f json` flag to show json format:

```
$ tapis apps show -f json tapis.app.imageclassify-1.0u3
```

```
{
  "id": "tapis.app.imageclassify-1.0u3",
  "name": "tapis.app.imageclassify",
  "version": "1.0",
  "revision": 3,
  "label": "Image Classifier",
  "lastModified": "6 months ago",
  "shortDescription": "Classify an image using a small ImageNet model",
  "longDescription": "",
  "owner": "cicsvc",
  "isPublic": true,
  "executionType": "CLI",
  "executionSystem": "tapis.execution.system",
  "deploymentSystem": "docking.storage",
  "available": true,
  "parallelism": "SERIAL",
  "defaultProcessorsPerNode": 1,
  "defaultMemoryPerNode": 1,
  "defaultNodeCount": 1,
  "defaultMaxRunTime": null,
  "defaultQueue": null,
  "tags": [
```

(continues on next page)

(continued from previous page)

```
    "tensorflow",
    "ImageNet"
  ],
  "ontology": [],
  "helpURI": "",
  "deploymentPath": "/home/docking/api/v2/prod/apps/tapis.app.imageclassify-1.0u3.zip
↪",
  "templatePath": "wrapper.sh",
  "testPath": "test/test.sh",
  "checkpointable": false,
  "modules": [
    "load tacc-singularity/2.6.0"
  ],
  "inputs": [],
  "parameters": [
    {
      "id": "imagefile",
      "value": {
        "visible": true,
        "required": true,
        "type": "string",
        "order": 0,
        "enquote": false,
        "default": "https://texassports.com/images/2015/10/16/bevo_1000.jpg",
        "validator": null
      },
      "details": {
        "label": "Image to classify",
        "description": "",
        "argument": "--image_file ",
        "showArgument": true,
        "repeatArgument": false
      },
      "semantics": {
        "minCardinality": 1,
        "maxCardinality": 1,
        "ontology": [
          "http://edamontology.org/format_3547"
        ]
      }
    },
    {
      "id": "predictions",
      "value": {
        "visible": true,
        "required": true,
        "type": "number",
        "order": 0,
        "enquote": false,
        "default": 5,
        "validator": null
      },
      "details": {
        "label": "Number of predictions to return",
        "description": null,
        "argument": "--num_top_predictions ",
        "showArgument": true,
```

(continues on next page)

(continued from previous page)

```
        "repeatArgument": false
      },
      "semantics": {
        "minCardinality": 1,
        "maxCardinality": 1,
        "ontology": []
      }
    }
  ],
  "outputs": [],
  "uuid": "3162334876895875561-242ac119-0001-005",
  "icon": null,
  "_links": {
    "self": {
      "href": "https://api.tacc.utexas.edu/apps/v2/tapis.app.imageclassify-1.0u3"
    },
    "executionSystem": {
      "href": "https://api.tacc.utexas.edu/systems/v2/tapis.execution.system"
    },
    "storageSystem": {
      "href": "https://api.tacc.utexas.edu/systems/v2/docking.storage"
    },
    "history": {
      "href": "https://api.tacc.utexas.edu/apps/v2/tapis.app.imageclassify-1.0u3/
↪history"
    },
    "metadata": {
      "href": "https://api.tacc.utexas.edu/meta/v2/data/?q=%7B%22associationIds%22%3A
↪%223162334876895875561-242ac119-0001-005%22%7D"
    },
    "owner": {
      "href": "https://api.tacc.utexas.edu/profiles/v2/cicsvc"
    },
    "permissions": {
      "href": "https://api.tacc.utexas.edu/apps/v2/tapis.app.imageclassify-1.0u3/pems"
    }
  }
}
```

8.4 Important Application Sections

Metadata: The metadata of the app json includes information about the app availability, runtime resources required, description, and much more. Some key information in the metadata section includes the identity of the HPC system (`executionSystem`) on which the app runs. In the above case, it is `tapis.execution.system`. Also, the `shortDescription` of the above app suggests that the function is to classify an image using a small ImageNet model.

Inputs: The above app does not contain any inputs. This section is used to describe required data and/or folders for running the app. Any files or folders specified in the inputs section will be staged to the execution system prior to running.

Parameters: This section describes important information, typically command line options, for running the app. The above app requires two parameters - a URL pointing to an image for the classifier, and the number of predictions that it should return.

Outputs: The above app does not define any outputs. This section may be used to specify expected output file or folder names, counts, and ontologies. While this feature is still under development, it can be used to aid in chaining apps together by providing the output of an app as input into a different app.

More information on each of these sections and understanding Tapis apps can be found in the Tapis Documentation.

CHAPTER 9

Prepare and Submit a Job

Continuing with the previous example of the Image Classifier app (see Find an Application), we know there are two parameters we need to specify: a URL pointing to an image for the classifier, and the number of predictions the app should return.

9.1 Build a Job Template File

To run an instance of this application (called a "job"), we first must assemble a json description of the job we would like to run. The simplest way to do this is to use the `tapis jobs init` command:

```
$ tapis jobs init tapis.app.imageclassify-1.0u3
```

```
{
  "name": "tapis.app.imageclassify-job-1585230186004",
  "appId": "tapis.app.imageclassify-1.0u3",
  "batchQueue": "normal",
  "maxRunTime": "01:00:00",
  "memoryPerNode": "1GB",
  "nodeCount": 1,
  "processorsPerNode": 1,
  "archive": true,
  "inputs": {},
  "parameters": {
    "imagefile": "https://texassports.com/images/2015/10/16/bevo_1000.jpg",
    "predictions": 5
  },
  "notifications": [
    {
      "event": "*",
      "persistent": true,
      "url": "taccuser@gmail.com"
    }
  ]
```

The only option given to the command is the name of the app. The output is a json template for submitting a job against the app. Metadata include a `name` for the job (which can be changed), the `appId`, and other runtime options. There are also the two parameters we expect filled with default options.

The only change we want to set now is `"archive": false` (see below for details on archiving jobs), and we also need to redirect this json into a file in order to submit a job. Execute:

```
$ tapis jobs init --no-archive --output job.json tapis.app.imageclassify-1.0u3
```

Tip: Use `tapis jobs init --help` to see further options

9.2 Submit a Job

Once you are satisfied that job.json contains the desired content, use the `tapis jobs submit` command to run an instance of this job:

```
$ tapis jobs submit -F job.json
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | f0cb69a1-63a4-4970-9921-843968e66723-007  |
| name   | tapis.app.imageclassify-job-1589290511905 |
| status | ACCEPTED                                 |
+--------+------------------------------------------+
```

An ACCEPTED status indicates the job.json was valid, and e-mail alerts (if they were specified in job.json) will track the progress of the job. Also take note of the long hexadecimal `id` when you submit the job. This identifier can be used to track progress and download results.

Note: If you are receiving too many e-mail alerts, try changing the job.json to only send an alert on `"event": "FINISHED"`

9.3 Track a Job

The Tapis CLI offers several commands to find and track the progress of jobs. If you know the specific job ID you are looking for, that can usually be passed to one of the jobs commands. Otherwise, you can list all jobs and find what you are looking for based on the name of the job and how recently it was run:

```
# List all jobs you have run
$ tapis jobs list
+------------------------------------------+-----------------------------------------
↪-+----------+
| id                                       | name                                    ␣
↪ | status   |
+------------------------------------------+-----------------------------------------
↪-+----------+
| f0cb69a1-63a4-4970-9921-843968e66723-007 | tapis.app.imageclassify-job-
↪1589290511905 | FINISHED |
+------------------------------------------+-----------------------------------------
↪-+----------+
```

(continues on next page)

(continued from previous page)

```
# List the status of a specific job
$ tapis jobs status f0cb69a1-63a4-4970-9921-843968e66723-007
+--------+-------------------------------------------+
| Field  | Value                                     |
+--------+-------------------------------------------+
| id     | f0cb69a1-63a4-4970-9921-843968e66723-007  |
| name   | tapis.app.imageclassify-job-1589290511905 |
| status | FINISHED                                  |
+--------+-------------------------------------------+

# List the history of a specific job
$ tapis jobs history f0cb69a1-63a4-4970-9921-843968e66723-007
+-------------------+---------------+--------------------------------------------------
↪-------------------------------------------------------------------------------------
↪----------------------------------------+
| status            | created       | description                                    ␣
↪                                                                                    ␣
↪                                         |
+-------------------+---------------+--------------------------------------------------
↪-------------------------------------------------------------------------------------
↪----------------------------------------+
| PENDING           | 3 minutes ago | Job processing beginning                       ␣
↪                                                                                    ␣
↪                                         |
| PROCESSING_INPUTS | 3 minutes ago | Identifying input files for staging            ␣
↪                                                                                    ␣
↪                                         |
| STAGED            | 3 minutes ago | Job inputs staged to execution system          ␣
↪                                                                                    ␣
↪                                         |
| STAGING_JOB       | 3 minutes ago | Staging runtime assets to execution system     ␣
↪                                                                                    ␣
↪                                         |
| STAGING_JOB       | 3 minutes ago | Fetching application assets from agave://
↪docking.storage//home/docking/api/v2/prod/apps/tapis.app.imageclassify-1.0u3.zip    ␣
↪                                                |
| STAGING_JOB       | 3 minutes ago | Staging runtime assets to agave://tapis.
↪execution.system//home/demo/scratch/taccuser/job-f0cb69a1-63a4-4970-9921-
↪843968e66723-007-tapis-app-imageclassify-job-1589290511905 |
| SUBMITTING        | 2 minutes ago | Submitting job to execution system             ␣
↪                                                                                    ␣
↪                                         |
| QUEUED            | 2 minutes ago | Job queued to execution system queue           ␣
↪                                                                                    ␣
↪                                         |
| RUNNING           | 2 minutes ago | Job running on execution system                ␣
↪                                                                                    ␣
↪                                         |
| CLEANING_UP       | a minute ago  | Job completed execution                        ␣
↪                                                                                    ␣
↪                                         |
| FINISHED          | a minute ago  | Job completed successfully                     ␣
↪                                                                                    ␣
↪                                         |
+-------------------+---------------+--------------------------------------------------
↪-------------------------------------------------------------------------------------
↪----------------------------------------+
```

(continues on next page)

(continued from previous page)

```
```

9.4 Download the Results

Once the job status is **FINISHED**, you can list what output is available:

```
$ tapis jobs outputs list f0cb69a1-63a4-4970-9921-843968e66723-007
+---------------------------------------------------+---------------+-----------+
| name                                              | lastModified  |    length |
+---------------------------------------------------+---------------+-----------+
| classifier_img.sif                                | 3 minutes ago | 379068416 |
| image.jpg                                         | 3 minutes ago |    116625 |
| predictions.txt                                   | 2 minutes ago |    498600 |
| tapis-app-imageclassify-job-1589290511905.err     | 2 minutes ago |       866 |
| tapis-app-imageclassify-job-1589290511905.ipcexe | 3 minutes ago |      2353 |
| tapis-app-imageclassify-job-1589290511905.out     | 3 minutes ago |         0 |
| tapis-app-imageclassify-job-1589290511905.pid     | 3 minutes ago |         6 |
| test                                              | 3 minutes ago |        21 |
| wrapper.sh                                        | 3 minutes ago |       196 |
+---------------------------------------------------+---------------+-----------+
```

For this app, there are several assets available to download. The important output is the predictions.txt file. You can choose to download all of the assets as a bundle, or a single file:

```
# Download a single file
$ tapis jobs outputs download f0cb69a1-63a4-4970-9921-843968e66723-007 predictions.txt
+-------------+-------+
| Field       | Value |
+-------------+-------+
| downloaded  | 1     |
| skipped     | 0     |
| messages    | 0     |
| elapsed_sec | 3     |
+-------------+-------+

 # Download all outputs
 $ tapis jobs outputs download --progress f0cb69a1-63a4-4970-9921-843968e66723-007
 Walking remote resource...
 Found 11 file(s) in 2s
 Downloading .agave.archive...
 Downloading .agave.log...
 Downloading classifier_img.sif...
 Downloading image.jpg...
 Downloading predictions.txt...
 Downloading tapis-app-imageclassify-job-1589290511905.err...
 Downloading tapis-app-imageclassify-job-1589290511905.ipcexe...
 Downloading tapis-app-imageclassify-job-1589290511905.out...
 Downloading tapis-app-imageclassify-job-1589290511905.pid...
 Downloading test.sh...
 Downloading wrapper.sh...
 Downloaded 11 files in 176s
 +-------------+-------+
 | Field       | Value |
 +-------------+-------+
```

(continues on next page)

(continued from previous page)

```
| downloaded  | 11    |
| skipped     | 0     |
| messages    | 0     |
| elapsed_sec | 178   |
+-------------+-------+
```

Note: The `--progress` flag prints progress of the download to STDOUT

9.5 Job Archives

Job archiving can either be set to true or false. If false, then the job outputs will remain in the execution folder created for your job. This may be on a scratch file system with a purge policy, so the outputs may be available for only a limited time through the jobs service. This can be useful when a job generates intermediate files that are not needed with the final output. To set archive mode to false, the following line should appear in your job.json file:

If job archiving is set to true, then the job outputs are written to a storage system and path that you specify, and will always be available through the jobs service. Use the following lines in your job.json file:

If you omit the name of the archive path, it will choose a default path on your storage system with the job identifier in the path name (recommended).

CHAPTER 10

Create a Private System

Many tenants automatically provide their users with private storage and execution systems connecting to TACC resources, but some do not. This guide walks through the process of creating your own private systems. This can also be used to help you connect to non-TACC lab servers, cloud VMs, and clusters.

10.1 Gather Relevant Information

To register any system in Tapis, you need the hostname and login credentials. The preferred login credentials are username and SSH key pairs. Storage systems (for storing files) require a default path where you have write access. Execution systems (for running jobs) also require a default path where job runtime files will be staged and the job will be executed. If it is an 'HPC' type execution system, then you also need information about the queueing system (queue names, limits, etc.).

For this demonstration, we will set up a storage system to access the TACC work directory (via Stampede2):

```
hostname: stampede2.tacc.utexas.edu
username: taccuser
credentials: <ssh keys>
storage path: /work/01234/taccuser
```

And we will set up an execution system for the Stampede2 HPC cluster:

```
hostname: stampede2.tacc.utexas.edu
username: taccuser
credentials: <ssh keys preferred>
storage path: /work/01234/taccuser
job runtime path: /scratch/01234/taccuser
queue_type: SLURM
queue: normal (limits in Stampede2 user guide)
```

Note: This guide assumes you have the appropriate permissions and credentials to access Stampede2. These can be attained by having an active Stampede2 Allocation

10.2 Register a Storage System

To register a system, you need to assemble a json description of the above requirements and some additional metadata. Start by saving the following storage system template in a file called `tacc.work.taccuser.json`:

```
{
  "id": "tacc.work.taccuser",
  "name": "Storage system for the TACC WORK directory",
  "description": "Storage system for the TACC WORK directory via Stampede2",
  "type": "STORAGE",
  "storage": {
    "host": "stampede2.tacc.utexas.edu",
    "port": 22,
    "protocol": "SFTP",
    "rootDir": "/work/01234/taccuser",
    "homeDir": "/",
    "auth":{
      "username":"taccuser",
      "publicKey": " <enter public key here> ",
      "privateKey": " <enter private key here> ",
      "type": "SSHKEYS"
    }
  }
}
```

Most fields are fairly self explanatory, but here is a brief breakdown of the important options:

- `id`: a unique identifier and how the system name appears for `tapis systems list` commands
- `name`: common display name for the system
- `description`: optional long plain text description of the system
- `type`: can be `STORAGE` or `EXECUTION`
- `host`: IP address or hostname of system
- `rootDir`: path of the virtual root directory on the remote system
- `homeDir`: path relative to `rootDir` for `tapis files -` operations
- `username`: this is your username for the target system
- `publicKey`: cut and paste your public key here
- `privateKey`: cut and paste your private key here

Edit the username and paths in the above template to match your username and work folder. A copy of your public key should be added to the `~/.ssh/authorized_keys` file on the remote host. The public and private key should be pasted on one line each similar to the following. Replace the line breaks in the private key with the newline character `\n`:

```
{
  "auth":{
    "username": "taccuser",
    "publicKey": "ssh-rsa␣
↪AAAAB3NzaC1yc2EBBAADAQABMQRgQqSuJdTi+VwMif8qouSSEWVduKZHpzOnS1zlknAyYXmQQFcaJ+vNAQayVMTqv+A+1lzxppT
↪IEMPzDuKb7F0qNFiH9m+OZJDYdIWS1rlN1oK32jHUi0xV8kM3KOLf2TIjDBUyZRpMGyQ= user@email.com
↪",
```

(continues on next page)

(continued from previous page)

```
    "privateKey": "-----BEGIN RSA PRIVATE KEY-----
↪\nMIIEpAIBAAKCAQEA1Jhi5BNiogg3NtALJepyTz5xS3j/
↪dpYBGf5ERBH0C\n4SCb9VAxOCyb4l+QDrOQnLRX2RV4JjHlw7r8qmc6IvPmk83oTYqYN2NuzMjxI\nsqjVfmJgnF4sPuQy+Pio
↪n9IDr\nTY2r/
↪B16XtzcjGYvhW35Avy2FlTHvJldwaxmY4UuNey7r9LXAved4nqTj7d\n5PVKgWB8Bu6h5U1EGgnPhFFi8MJCO4/
↪bByqAYdEffC9Y+cWBFq749XNPafid\nDlKFza44RR5Fg86OZxJW7NGoMnIjVYRIcUQIDAQABAoIBAQCovogIBscMW6R/
↪\nfTwM/h3OlUu9EdlVOfygwkq5GfdbPBco291UOmDwN08aryTR8JtVLPO5ZtevX\nTVXVpWtejdLr5aL/R/
↪uYxhxaIoeI7ppQBk3daSNsZia2lRp1j4qil\nyKfy5WxHdzjAhg3gamYtTk981qJIOSR0kQxxz3ax23BN5C/
↪r1uqHK1hFUlCgx\nRrjt2M2/TvFtGZdRmxH4Kdco7IeOtj0xAYS/hGBV4CRa+4zWb3ikNOVxcFL61\nuT/
↪60043DsVI22B5zv3XODtfSjquqlYl5eHZdf+HdL6u91CKrjmvpg80OfQ\ngmPwhOjdAoGBAOtRhXta/
↪Y8X1U+XykaXfVFsfzFsslMtI73XII+nKYdtDFlSl\nLYg6PB5Gk9Q++RdinHzL7DwAXOVWW2nwfoEKxjsYCw4ihYVO/
↪UEG\nqqCeu0X/r9N6u78HfeZEX5XH3+QtR/
↪d9bP2mLjhY8LTAoGBAOdH\nnHnrMpeiEzou+5UC3lKRUN84LX/
↪o9kp6t6WSF5oT7tQEyJKVICgLBOMVbASvXZxncYziYJKIzrqDkC+QXdZpF0x/
↪u04vryDz9ySl9rhBYaD74e+FFXkDImMAQ\nCL1InIelCmXcWsORJd+5yCGOSS3TL2lA+1YXLAoGf47hMm/
↪uT0HvzVhDq7\nD+764ZgRHjN8tpn9N0hz/
↪Gj0zaw+9lOXEXG1DnlGzo016sAOc+2tFZx\n3j8w9cZQJ0zTE2u7Lz8CL9yKXicsOgFhdeyrF4AwtJ2CLtZF383wim2QFi4/
↪Ypkl\nL4lsYnJYnJjQCKgA6bROu0+rA1TUvCzXHbgH7t6eYRcZeKnJNZ+m3PhBs+8W\nov4nLLTz8Q7GN8g6T1/
↪/QojS8y  ␣
↪ZR9GAr0Z0BbtW8om+fVehPFAMm8x6tS4sTFl0\nUp+i0r4VF7PnvfSIC+AHJUe+a4XPmmphVsnxEpsS+tQ2yUh7Akmt\np8WOE
↪Xr8fZrz8KpmSehT99a+QY6gkIoWrfQ5xS7g6\nI/GDX2x54eANWX0xXKMQXfTU+WN6s5WPl/BL+/
↪Cj43Hfg==\n-----END RSA PRIVATE KEY-----",
    "type": "SSHKEYS"
  }
}
```

Warning: Remember, never share this json file because it contains a plain text copy of your private key.

You will need to keep a copy of this file to edit the storage system in the future. To register this system with Tapis, use the following command:

```
$ tapis systems create -F tacc.work.taccuser.json
+-----------------------+-------------------------------------------------------+
| Field                 | Value                                                 |
+-----------------------+-------------------------------------------------------+
| id                    | tacc.work.taccuser                                    |
| name                  | Storage system for TACC work directory                |
| type                  | STORAGE                                               |
| default               | False                                                 |
| available             | True                                                  |
| description           | Storage system for TACC work directory via Stampede2 |
| executionType         | None                                                  |
| globalDefault         | False                                                 |
| lastModified          | just now                                              |
| maxSystemJobs         | None                                                  |
| maxSystemJobsPerUser  | None                                                  |
| owner                 | taccuser                                              |
| public                | False                                                 |
| revision              | 1                                                     |
| scheduler             | None                                                  |
| scratchDir            | None                                                  |
| site                  | None                                                  |
| status                | UP                                                    |
| uuid                  | 383424038079107562-242ac112-0001-006                  |
```

(continues on next page)

(continued from previous page)

```
| workDir               | None                                                     |
+-----------------------+----------------------------------------------------------+
```

Confirm that it worked by searching for the storage system and listing files in the root directory:

```
$ tapis systems search --id eq tacc.work.taccuser
+---------------------+-----------------------------------------+----------+---------+
| id                  | name                                    | type     | default |
+---------------------+-----------------------------------------+----------+---------+
| tacc.work.taccuser  | Storage system for TACC work directory | STORAGE  | False   |
+---------------------+-----------------------------------------+----------+---------+

$ tapis files list agave://tacc.work.taccuser/
+-----------+--------------+--------+
| name      | lastModified | length |
+-----------+--------------+--------+
| jobs      | 2 years ago  |   4096 |
| maverick  | 2 years ago  |   4096 |
| stampede2 | 2 years ago  |   4096 |
| wrangler  | 2 years ago  |   4096 |
+-----------+--------------+--------+
```

10.3 Register an Execution System

An execution system contains many of the same fields as a storage system, but it is a bit more involved. Save the following template for a Stampede2 execution system into a file called `tacc.stampede2.taccuser`:

```
{
  "id": "tacc.stampede2.taccuser",
  "name": "Execution system for TACC Stampede2",
  "description": "Execution system for TACC Stampede2",
  "type": "EXECUTION",
  "executionType": "HPC",
  "scheduler": "SLURM",
  "maxSystemJobsPerUser": 50,
  "scratchDir": "/scratch/01234/taccuser",
  "login": {
    "host": "stampede2.tacc.utexas.edu",
    "port": 22,
    "protocol": "SSH",
    "auth": {
      "username": "taccuser",
      "publicKey": " <enter public key here> ",
      "privateKey": " <enter private key here> ",
      "type": "SSHKEYS"
    }
  },
  "storage": {
    "host": "stampede2.tacc.utexas.edu",
    "port": 22,
    "protocol": "SFTP",
    "rootDir": "/",
    "homeDir": "/work/01234/taccuser",
    "auth": {
```

(continues on next page)

(continued from previous page)

```
      "username": "taccuser",
      "publicKey": " <enter public key here> ",
      "privateKey": " <enter private key here> ",
      "type": "SSHKEYS"
    }
  },
  "queues": [
    {
      "name": "normal",
      "maxProcessorsPerNode": 68,
      "maxMemoryPerNode": "96GB",
      "maxNodes": 256,
      "maxRequestedTime": "48:00:00",
      "customDirectives": "-A <enter allocation name here>",
      "default": true
    }
  ]
}
```

Some of the new or changed fields in this execution system include:

- `type`: execution system rather than storage system
- `executionType`: `HPC` indicates a cluster with a job scheduler
- `scheduler`: Stampede2 uses a SLURM scheduler
- `maxSystemJobsPerUser`: maximum concurrent jobs on the system per user
- `scratchDir`: path for job working directory at runtime, relative to `rootDir`
- `login`: similar to *storage*, host and credential information
- `queues`: an array of batch queue definitions for the HPC system

For this execution system, there are two locations to cut and paste your SSH keys. Again, because keys will be stored in plain text in this file, do not share this file with anyone and keep it secure. In addition, the `queues` parameter has an option called `customDirectives` which should contain the name of an allocation you have access to on Stampede2. And finally, as before, make sure to change the username and paths to match your account on the HPC system.

Once the appropriate changes have been made to the json file, register the system with Tapis using the following command:

```
$ tapis systems create -F tacc.stampede2.taccuser.json
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| id                   | tacc.stampede2.taccuser              |
| name                 | Execution system for TACC Stampede2  |
| type                 | EXECUTION                            |
| default              | False                                |
| available            | True                                 |
| description          | Execution system for TACC Stampede2  |
| executionType        | HPC                                  |
| globalDefault        | False                                |
| lastModified         | just now                             |
| maxSystemJobs        | 2147483647                           |
| maxSystemJobsPerUser | 50                                   |
```

(continues on next page)

(continued from previous page)

```
| owner                 | taccuser                             |
| public                | False                                |
| revision              | 1                                    |
| scheduler             | SLURM                                |
| scratchDir            | /scratch/01234/taccuser/             |
| site                  | None                                 |
| status                | UP                                   |
| uuid                  | 4903282542648684054-242ac112-0001-006 |
| workDir               |                                      |
+-----------------------+--------------------------------------+
```

Finally, confirm that the system exists by searching for it then listing the available queues:

```
# Search for your private systems
$ tapis systems search --public eq false
+-------------------------+-----------------------------------------+-----------+------
↪---+
| id                      | name                                    | type      |␣
↪default |
+-------------------------+-----------------------------------------+-----------+------
↪---+
| tacc.stampede2.taccuser | Execution system for TACC Stampede2     | EXECUTION |␣
↪False   |
| tacc.work.taccuser      | Storage system for TACC work directory | STORAGE   |␣
↪False   |
+-------------------------+-----------------------------------------+-----------+------
↪---+

# List queues on the execution system
$ tapis systems queues list -f json tacc.stampede2.taccuser
[
  {
    "name": "normal",
    "description": null,
    "default": true,
    "maxUserJobs": -1,
    "maxRequestedTime": "48:00:00"
  }
]
```

10.4 Additional Help

Further information about creating storage and execution systems, including full descriptions of the parameters above as well as other optional parameters, can be found in the Tapis platform documentation

CHAPTER 11

Modify an Existing System

Tapis system definitions (written in `json` file format) can be added to a tenant using the command line interface. To modify a system after it has been added, you must edit the original `json` file and use the CLI to submit the change.

11.1 Modify a Storage System

In a previous section of this user guide, we registered a new storage system called `tacc.work.taccuser` using the following `json`, which was stored in a file called `tacc.work.taccuser.json`:

```
{
  "id": "tacc.work.taccuser",
  "name": "Storage system for the TACC WORK directory",
  "description": "Storage system for the TACC WORK directory via Stampede2",
  "type": "STORAGE",
  "storage": {
    "host": "stampede2.tacc.utexas.edu",
    "port": 22,
    "protocol": "SFTP",
    "rootDir": "/work/01234/taccuser",
    "homeDir": "/",
    "auth":{
      "username":"taccuser",
      "publicKey": " <enter public key here> ",
      "privateKey": " <enter private key here> ",
      "type": "SSHKEYS"
    }
  }
}
```

If you need to change the hostname, paths, ssh keys, or any other field (other than the `id`, which is immutable), the appropriate method would be to edit the above file to reflect the change, then use the Tapis CLI to edit the existing storage system. For the purposes of this example, we may want to change the plain text `name` parameter to include more detail. Modify the json file and submit the changes as follows:

```
$ tapis systems update -F tacc.work.taccuser.json tacc.work.taccuser
+-----------------------+----------------------------------------------------------+
| Field                 | Value                                                    |
+-----------------------+----------------------------------------------------------+
| id                    | tacc.work.taccuser                                       |
| name                  | Storage system for the TACC work directory via Stampede2 |
| type                  | STORAGE                                                  |
| default               | False                                                    |
| available             | True                                                     |
| description           | Storage system for the TACC WORK directory via Stampede2 |
| executionType         | None                                                     |
| globalDefault         | False                                                    |
| lastModified          | just now                                                 |
| maxSystemJobs         | None                                                     |
| maxSystemJobsPerUser  | None                                                     |
| owner                 | taccuser                                                 |
| public                | False                                                    |
| revision              | 2                                                        |
| scheduler             | None                                                     |
| scratchDir            | None                                                     |
| site                  | None                                                     |
| status                | UP                                                       |
| uuid                  | 383424038079107562-242ac112-0001-006                     |
| workDir               | None                                                     |
+-----------------------+----------------------------------------------------------+
```

The plain text response should include the new value for the `name` parameter. You can also use the `tapis systems history` command to check that the update was accepted:

```
$ tapis systems history tacc.work.taccuser
+-------------+----------------------+-------------------------------------------------
↪-----+
| status      | created              | description                                     ␣
↪     |
+-------------+----------------------+-------------------------------------------------
↪-----+
| CREATED     | 2020-05-12T04:57:03Z | This system was created                         ␣
↪     |
| ROLES_GRANT | 2020-05-12T13:16:48Z | User jdoe was granted the role of GUEST by␣
↪taccuser |
| UPDATED     | 2020-05-12T14:09:18Z | This system was updated                         ␣
↪     |
+-------------+----------------------+-------------------------------------------------
↪-----+
```

11.2 Modify an Execution System

In a previous section we registered a new execution system for the Stampede2 HPC cluster. In our system description, we only included one queue (`normal`), although Stampede2 has many more queues available. To add an additional queue, return to the original json file called `tacc.stampede2.taccuser.json` and add another json object to the queue array:

Save that new file and update the existing system with the following:

```
$ tapis systems update -F tacc.stampede2.taccuser.json tacc.stampede2.taccuser
+----------------------+---------------------------------------+
| Field                | Value                                 |
+----------------------+---------------------------------------+
| available            | True                                  |
| default              | False                                 |
| description          | Execution system for TACC Stampede2   |
| executionType        | HPC                                   |
| globalDefault        | False                                 |
| id                   | tacc.stampede2.taccuser               |
| lastModified         | just now                              |
| maxSystemJobs        | 2147483647                            |
| maxSystemJobsPerUser | 50                                    |
| name                 | Execution system for TACC Stampede2   |
| owner                | taccuser                              |
| public               | False                                 |
| revision             | 2                                     |
| scheduler            | SLURM                                 |
| scratchDir           | /scratch/01234/taccuser/              |
| site                 | None                                  |
| status               | UP                                    |
| type                 | EXECUTION                             |
| uuid                 | 5042654862881657322-242ac113-0001-006 |
| workDir              |                                       |
+----------------------+---------------------------------------+

$ tapis systems queues list tacc.stampede2.taccuser
+------------+-------------+---------+-------------+------------------+
| name       | description | default | maxUserJobs | maxRequestedTime |
+------------+-------------+---------+-------------+------------------+
| skx-normal | None        | True    |          -1 | 48:00:00         |
| normal     | None        | False   |          -1 | 48:00:00         |
+------------+-------------+---------+-------------+------------------+
```

CHAPTER 12

Create a Custom App

You can find Tapis applications ("apps") in your tenant's catalog by using the `tapis apps list` command described previously. If the app you are looking for is not available, you can create your own and add it to the catalog.

12.1 Components of an App

The essential components you need to create your own app are:

1. An app bundle directory containing definitions and assets for the app
2. A Docker image containing the executable and all runtime dependencies
3. A wrapper script (written in bash) that runs the executable

12.2 Create an App by Example

The best way to demonstrate the creation of a custom app is by example. The following sub-pages will go through the process:

12.2.1 Initialize the App Directory

At the core of a Tapis app is an executable. Going in to the app building process, it is generally assumed that the developer has an executable in mind and the knowledge to run an instance of the executable with given inputs and / or parameters. In this example, we will create an app for the FastQC tool. FastQC is a publicly-available quality control tool for raw next-gen sequencing data.

Structure of an App

To begin, run the command `tapis apps init` and give an arbitrary name for a test app:

```
$ tapis apps init test_app
+--------+--------------------------------+
| stage  | message                        |
+--------+--------------------------------+
| setup  | Project name: test_app         |
| setup  | Safened project name: test_app |
| setup  | Project path: ./test_app       |
| clone  | Project path: ./test_app       |
+--------+--------------------------------+
```

This will create a new template app folder (in this case, called `test_app/`) with the following form:

```
$ tree test_app/
test_app/
├── Dockerfile
├── app.json
├── assets
│   ├── lib
│   │   ├── VERSION
│   │   └── container_exec.sh
│   ├── runner.sh
│   └── tester.sh
├── job.json
└── project.ini
```

Several files and folders are created automatically. It is a good idea to take some time now to look through the directory tree and examine the contents of each file. A brief summary of the files are as follows:

- `Dockerfile`: a Dockerfile for the app runtime
- `app.json`: json file describing the app metadata, inputs, parameters, and outputs
- `VERSION`: version file containing the image tag
- `container_exec.sh`: utility script for executing a container on a TACC HPC system
- `runner.sh`: main run script for the app; takes input and parameters from app.json
- `tester.sh`: legacy script that may be used to run a local test
- `job.json`: template for a job json file specific to this app
- `project.ini`: initialization parameters for the app which are injected in to app.json

Initialize the FastQC App

Use the `tapis apps init` command again, but this time provide additional flags to indicate the name and version of the app:

```
$ tapis apps init --app-label fastqc --app-description "FastQC app" --app-version 0.
↪11.9 fastqc_app
+--------+-----------------------------------+
| stage  | message                           |
+--------+-----------------------------------+
| setup  | Project name: fastqc_app          |
| setup  | Project description: FastQC app   |
| setup  | Project version: 0.11.9           |
| setup  | Safened project name: fastqc_app  |
| setup  | Project path: ./fastqc_app        |
```

(continues on next page)

(continued from previous page)

```
| clone | Project path: ./fastqc_app       |
+-------+----------------------------------+

$ tree fastqc_app/
fastqc_app/
├── Dockerfile
├── app.json
├── assets
│   ├── lib
│   │   ├── VERSION
│   │   └── container_exec.sh
│   ├── runner.sh
│   └── tester.sh
├── job.json
└── project.ini
```

From here on, we will refer to the location of this app bundle as `~/fastqc_app/`. In the next sections, we will go through the template files one by one to customize them for this particular app.

~/fastqc_app/project.ini

The first file to examine is called `project.ini`, which contains initialization parameters for the app. By default, the fields are populated by some of the flags specified on the command line or picked up from the environment:

```
[app]
name = fastqc_app
label = fastqc_app
description = FastQC app
version = 0.11.9
; bundle = assets
; deployment_path =
deployment_system = tacc.work.taccuser
execution_system = tacc.stampede2.taccuser

[docker]
dockerfile = Dockerfile
namespace = taccuser
repo = fastqc_app
tag = 0.11.9

[env]

[git]
branch = master
; remote =

[grants]
; read =
; execute =
; update =

[job]
```

The parameters listed above will be interpreted and injected into the app when you deploy it. We need to make some changes to the data above. Set the following:

```
deployment_system = tacc.work.taccuser
execution_system = tacc.stampede2.taccuser
```

These should be the names of your private storage and execution systems, respectively.

`~/fastqc_app/app.json`

This is a templated app json file. By default, it will grab the app `name`, `version`, *executionSystem*, *deploymentSystem*, and other parameters from your `project.ini`. Now is a good time to modify this file if a typical job run against this app would require, e.g., more than one node or a non standard queue. The jinja2-formatted fields surrounded by double curly braces `{{ }}` are take from `app.ini`.

Specific to FastQC, one input is required - a fastq file. Modify `app.json` to expect one input fastq file as shown below:

```
{
  "checkpointable": false,
  "name": "{{ username }}-{{ app.name }}",
  "executionSystem": "{{ app.execution_system }}",
  "executionType": "HPC",
  "deploymentPath": "{{ username }}/apps/{{ app.name }}-{{ app.version }}",
  "deploymentSystem": "{{ app.deployment_system }}",
  "helpURI": "",
  "label": "{{ app.label }}",
  "shortDescription": "{{ app.description }}",
  "longDescription": "",
  "modules": [
    "load tacc-singularity"
  ],
  "ontology": [],
  "parallelism": "SERIAL",
  "tags": [],
  "templatePath": "runner.sh",
  "testPath": "tester.sh",
  "version": "{{ app.version }}",
  "defaultMaxRunTime": "00:30:00",
  "inputs":[
    {
      "id": "fastq",
      "value": {
        "default": "",
        "visible": true,
        "required": true
      },
      "semantics": {
        "ontology": [
          "http://edamontology.org/format_1930"
        ]
      },
      "details": {
        "label": "FASTQ sequence file"
      }
    }
  ],
  "parameters": [
    {
```

(continues on next page)

(continued from previous page)

```
      "id": "CONTAINER_IMAGE",
      "value": {
        "default": "{{ docker.namespace }}/{{ docker.repo }}:{{ docker.tag }}",
        "type": "string",
        "visible": false,
        "required": true,
        "order": 1000
      }
    }
  ],
  "outputs": []
}
```

Please refer back to the previous App Documentation for a detailed breakdown of a typical app json file.

~/fastqc_app/job.json

The `job.json` file contains minimal information. The only change needed at this time is to add the expect input:

```
{
  "name": "{{ app.name }}-test-{{ iso8601_basic_short }}",
  "appId": "{{ app.name }}-{{ app.version}}",
  "archive": false,
  "inputs": {
    "fastq": ""
  },
  "parameters": {}
}
```

Next Steps

If you have been following along, these files are ready to deploy for your app:

```
fastqc_app/
├── Dockerfile
├── app.json                    # Done
├── assets
│   ├── lib
│   │   ├── VERSION
│   │   └── container_exec.sh   # Do not modify
│   ├── runner.sh
│   └── tester.sh               # Do not modify
├── job.json                    # Done
└── project.ini                 # Done
```

Next, we will build the `Dockerfile` and `runner.sh`.

12.2.2 Containerize the Executable

If an image of your executable already exists, and was created by a trusted source, consider using that rather than building your own. You may find existing images on hubs such as Docker Hub, BioContainers, or Quay.io.

This tutorial is a quick and dirty summary of how to build your own Docker image as if there is not one available for your executable. This is not meant to replace the full Docker documentation.

We will continue with the example of FastQC from the previous page.

Choose a Source Image

Prerequisite: You should have a Docker ID and docker should be installed on your local machine.

The only dependency for FastQC is a reasonably recent Java Runtime Environment. Thus, most modern Linux OS-es should suffice. The TACC Docker hub organization provides a few base images that would work for this. Pull one manually now so it is in your environment:

```
$ docker pull tacc/tacc-ubuntu18-mvapich2.3-psm2
```

A list of available base images can be found here.

Also, now is a good time to prepare a `src/` directory in the application bundle. If your code is a python script, for example, this is where you want to put it. This directory is not necessary, but nice to have if you want to keep your executable together with the app assets:

```
$ cd ~/fastqc-app/
$ mkdir src/
```

Install and Test Interactively

The installation process for `fastqc` is extremely simple. It is good practice to test the installation interactively, and record the steps for a Dockerfile:

```
# Start an interactive docker session
$ docker run --rm -it tacc/tacc-ubuntu18-mvapich2.3-psm2

# Update and install necessary packages
[docker] $ apt-get update
[docker] $ apt-get upgrade -y
[docker] $ apt-get install wget -y
[docker] $ apt-get install zip -y
[docker] $ apt-get install default-jre -y

# Install FastQC
[docker] $ wget https://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.
↪11.9.zip
[docker] $ unzip fastqc_v0.11.9.zip
[docker] $ rm fastqc_v0.11.9.zip
[docker] $ chmod +x /FastQC/fastqc
```

After a bit of trial and error, the commands above are a reasonably short path to installing the *fastqc* executable. You can test it from within the docker image to make sure it is working by, for example:

```
[docker] $ /FastQC/fastqc -h

            FastQC - A high throughput sequence QC analysis tool

SYNOPSIS

        fastqc seqfile1 seqfile2 .. seqfileN

    fastqc [-o output dir] [--(no)extract] [-f fastq|bam|sam]
```

(continues on next page)

(continued from previous page)

```
          [-c contaminant file] seqfile1 .. seqfileN
... etc.
```

Note on Source Code and Mounting Directories

In this instance, we could have downloaded the source zip file for FastQC directly to the `src/` directory of our app bundle, then mounted that directory within the image, e.g.:

```
$ cd ~/fastqc-app/src/
$ wget https://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.9.zip
$ docker run --rm -it -v $PWD:/opt/src tacc/tacc-ubuntu18-mvapich2.3-psm2
... etc.
```

That route is perfectly reasonable and can be followed here. In fact, if your app is a standalone python script, for example, this is the best method for including it in your Docker image.

However, some packages have very large zip or tar.gz files (100s of MB), and would be cumbersome to keep in this fastqc app bundle folder. It is up to the app developer to find the balance between completeness of source files and responsible disk usage.

Here, we decide to not download the source permanently. Instead, we make a record of where the source came from. For example:

```
$ cd ~/fastqc-app/src/
$ echo "Source: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.
↪11.9.zip" \
    >> README.md
```

Write the Dockerfile

Next, translate the steps required to install your software package into a resonable `Dockerfile`. The `Dockerfile` should be located at the root directory, `~/fastqc-app/Dockerfile`:

```
FROM tacc/tacc-ubuntu18-mvapich2.3-psm2

RUN apt-get update \
    && apt-get upgrade -y \
    && apt-get install wget -y \
    && apt-get install zip -y \
    && apt-get install default-jre -y

RUN wget https://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.9.zip␣
↪\
    && unzip fastqc_v0.11.9.zip \
    && rm fastqc_v0.11.9.zip \
    && chmod +x FastQC/fastqc

ENV PATH "/FastQC/:$PATH"
```

Build and Test the Image

Navigate to the top of the app directory, `~/fastqc-app/`, and the command to build a new Docker image is:

```
$ docker build -f Dockerfile --force-rm -t taccuser/fastqc:0.11.9 ./
```

Once built, test the new image with an example command:

```
$ docker run --rm fastqc:0.11.9 fastqc -h
- or -
$ docker run --rm fastqc:0.11.9 perl /FastQC/fastqc -h
```

Note: Calling the complete path to executables is sometimes safer than relying on PATH environment variables

If you see the FastQC help text, the installation likely was successful. At this time, it might be prudent to test with real data as well. Download some test data into a *~/fastqc-app/tests/* directory:

```
$ cd ~/fastqc-app/
$ mkdir tests/

# Download random sample data or provide your own
# wget https://molb7621.github.io/workshop/_downloads/SP1.fq
```

Next, run the FastQC pipeline on the example data:

```
$ docker run -v $PWD:/data fastqc:0.11.9 perl /FastQC/fastqc /data/SP1.fq
```

If successful, you should find the output files `SP1_fastqc.html` and `SP1_fastqc.zip` in the `~/fastqc-app/tests/` directory.

Push Your Image to the Cloud

If you are happy with the tests, push your Docker image to a publicly available repository. It can be your own personal repository as long as it is set to public, and not private. To push to your own repository, make sure it was namespaced with your Docker ID. Then:

```
$ docker push taccuser/fastqc:0.11.9
```

Assemble Run Commands

The final step is to put instructions in `runner.sh` on how the app should be run. In general, these are the same commands we used for testing above.

Most of the lines in the default file should be left alone. See the last three lines of this file for what should be added:

```
# Allow over-ride
if [ -z "${CONTAINER_IMAGE}" ]
then
    version=$(cat ./_util/VERSION)
    CONTAINER_IMAGE="index.docker.io/taccuser/fastqc:${version}"
fi
. lib/container_exec.sh

# Write an excution command below that will run a script or binary inside the
# requested container, assuming that the current working directory is
# mounted in the container as its WORKDIR. In place of 'docker run'
# use 'container_exec' which will handle setup of the container on
```

(continues on next page)

(continued from previous page)

```
# a variety of host environments.
#
# Here is a template:
#
# container_exec ${CONTAINER_IMAGE} COMMAND OPTS INPUTS
#
# Here is an example of counting words in local file 'poems.txt',
# outputting to a file 'wc_out.txt'
#
# container_exec ${CONTAINER_IMAGE} wc poems.txt > wc_out.txt
#

# set -x

# set +x

COMMAND="perl /FastQC/fastqc"
PARAMS="${fastq}"
container_exec ${CONTAINER_IMAGE} ${COMMAND} ${PARAMS}
```

Update the `VERSION` File

Finally, put the version / docker tag into the file located at `~/fastqc_app/assets/lib/VERSION`:

```
$ echo "0.11.9" >> ~/fastqc_app/assets/lib/VERSION
$ cat ~/fastqc_app/assets/lib/VERSION
0.11.9
```

12.2.3 Deploy and Test the App

At this point, your app bundle directory structure should look similar to:

```
fastqc_app/
├── Dockerfile                  # Done
├── app.json                    # Done
├── assets
│   ├── lib
│   │   ├── VERSION             # Done
│   │   └── container_exec.sh   # Do not modify
│   ├── runner.sh               # Done
│   └── tester.sh               # Do not modify
├── job.json                    # Done
├── project.ini                 # Done
├── src
│   └── mycode.py               # Optional
└── test
    └── SP1.fq                  # Optional
```

The final steps are to deploy the app in to the catalog on the tenant, then run a test job to confirm it is working.

Upload Test Data

The Tapis app takes FASTQ data as input. Now is a good time to upload some sample data to a storage system.

```
# Make a directory for test data
$ tapis files mkdir agave://tacc.work.taccuser/ test-data
+--------------+---------------------------------------+
| Field        | Value                                 |
+--------------+---------------------------------------+
| name         | test-data                             |
| uuid         | 1105399583769563626-242ac112-0001-002 |
| owner        | taccuser                              |
| path         | /test-data                            |
| lastModified | 2020-05-13T18:40:00.027-05:00         |
| source       | None                                  |
| status       | STAGING_COMPLETED                     |
| nativeFormat | dir                                   |
| systemId     | tacc.work.taccuser                    |
+--------------+---------------------------------------+

# Upload the SP1.fq file
$ tapis files upload agave://tacc.work.taccuser/test-data/ ./test/SP1.fq
+-------------------+----------+
| Field             | Value    |
+-------------------+----------+
| uploaded          | 1        |
| skipped           | 0        |
| messages          | 0        |
| bytes_transferred | 21.94 kB |
| elapsed_sec       | 3        |
+-------------------+----------+

# Confirm the file exists on the Tapis storage system
$ tapis files list agave://tacc.work.taccuser/test-data/
+--------+--------------+--------+
| name   | lastModified | length |
+--------+--------------+--------+
| SP1.fq | 1 minute ago |  22471 |
+--------+--------------+--------+
```

The test data can be referenced in future calls to the app as:

```
agave://tacc.work.taccuser/test-data/SP1.fq
```

Deploy the App

The `tapis apps deploy` command does multiple things. First, it builds and pushes your docker image to the namespace specified in `project.ini`. Second, it uploads the app assets to a Tapis storage space specified by `deploymentSystem` and `deploymentPath` in the app json file. Finally, it registers the app with the tenant.

Navigate in to the app bundle folder and issue:

```
$ pwd
~/fastqc_app/

$ ls
Dockerfile      app.json        assets/         job.json
project.ini     src/            test/

$ tapis apps deploy -W ./
```

(continues on next page)

(continued from previous page)

```
+---------+-------------------------------------------------------------------------
↪-------------------------------------------------------------------------
↪-----------------------------+
| stage   | message                                                                  ␣
↪                                                                                   ␣
↪                              |
+---------+-------------------------------------------------------------------------
↪-------------------------------------------------------------------------
↪-----------------------------+
| build   | Step 1/4 : FROM tacc/tacc-ubuntu18-mvapich2.3-psm2                       ␣
↪                                                                                   ␣
↪                              |
| build   |  ---> d554e642ddc5                                                       ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
| build   | Step 2/4 : RUN apt-get update     && apt-get upgrade -y     && apt-get␣
↪install wget -y     && apt-get install zip -y     && apt-get install default-jre -y␣
↪                                  |
| build   |  ---> Using cache                                                        ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
| build   |  ---> aa1f50856b62                                                       ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
| build   | Step 3/4 : RUN wget https://www.bioinformatics.babraham.ac.uk/projects/
↪fastqc/fastqc_v0.11.9.zip     && unzip fastqc_v0.11.9.zip     && rm fastqc_v0.11.9.
↪zip     && chmod +x FastQC/fastqc |
| build   |  ---> Using cache                                                        ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
| build   |  ---> 3bb6917b68d6                                                       ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
| build   | Step 4/4 : ENV PATH "/FastQC/:$PATH"                                     ␣
↪                                                                                   ␣
↪                              |
| build   |  ---> Using cache                                                        ␣
↪                                                                                   ␣
↪                              |
|         |                                                                          ␣
↪                                                                                   ␣
↪                              |
```

(continues on next page)

(continued from previous page)

```
| build  |  ---> 356927b0a8f6                                                 ␣
↪                                                                             ␣
↪                              |
|        |                                                                    ␣
↪                                                                             ␣
↪                              |
| build  | Successfully built 356927b0a8f6                                    ␣
↪                                                                             ␣
↪                              |
|        |                                                                    ␣
↪                                                                             ␣
↪                              |
| build  | Successfully tagged taccuser/fastqc_app:0.11.9                     ␣
↪                                                                             ␣
↪                              |
|        |                                                                    ␣
↪                                                                             ␣
↪                              |
| push   | The push refers to repository [docker.io/taccuser/fastqc_app]      ␣
↪                                                                             ␣
↪                              |
| push   | 0.11.9: digest:␣
↪sha256:29eb2fdb1503fdd38ae311dabfc13958f0910253580614dba0d3ac2dd0753e41 size: 4085 ␣
↪                                                                             ␣
↪     |
| upload | assets/runner.sh                                                   ␣
↪                                                                             ␣
↪                              |
| upload | assets/tester.sh                                                   ␣
↪                                                                             ␣
↪                              |
| upload | assets/lib/VERSION                                                 ␣
↪                                                                             ␣
↪                              |
| upload | assets/lib/container_exec.sh                                       ␣
↪                                                                             ␣
↪                              |
| create | Created Tapis app fastqc_app-0.11.9 revision 1                     ␣
↪                                                                             ␣
↪                              |
+--------+--------------------------------------------------------------------------
↪---------------------------------------------------------------------------------
↪----------------------------+
```

If all goes well, you should see a successful message at the end of the log above, and you should see the new app listed in the apps catalog:

```
$ tapis apps search --name like fastqc
+----------------------------+----------+------------+-------------------+----------+--
↪-----------------------+
| id                         | revision | label      | shortDescription  | isPublic |␣
↪executionSystem          |
+----------------------------+----------+------------+-------------------+----------+--
↪-----------------------+
| taccuser-fastqc_app-0.11.9 |        2 | fastqc_app | FastQC app        | False    |␣
↪tacc.stampede2.taccuser |
```

(continues on next page)

(continued from previous page)

```
+----------------------------+----------+------------+------------------+----------+--
↪------------------------+
```

Submit a Test Job

Submitting a test job has been described previously in this how-to guide. Here, testing will be performed in the same way. First, create an appropriate `job.json` file.

```
$ tapis jobs init --no-archive --output fastqc_job.json taccuser-fastqc_app-0.11.9
```

Which will output the following json, which can be streamed into a file for submission (add the highlighted test data line):

```
{
  "name": "taccuser-fastqc_app-job-1589377205989",
  "appId": "taccuser-fastqc_app-0.11.9",
  "batchQueue": "skx-normal",
  "maxRunTime": "01:00:00",
  "memoryPerNode": "1GB",
  "nodeCount": 1,
  "processorsPerNode": 1,
  "archive": false,
  "inputs": {
    "fastq": "agave://tacc.work.taccuser/test-data/SP1.fq"
  },
  "parameters": {},
  "notifications": [
    {
      "event": "FINISHED",
      "persistent": true,
      "url": "taccuser@gmail.com"
    },
    {
      "event": "FAILED",
      "persistent": true,
      "url": "taccuser@gmail.com"
    }
  ]
}
```

Then, submit the test job:

```
$ tapis jobs submit -F fastqc_job.json
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | 4e972f77-5bf9-446e-87a2-3541c4ea5745-007 |
| name   | taccuser-fastqc_app-job-1589377205989    |
| status | ACCEPTED                                 |
+--------+------------------------------------------+
```

Finally, when the job status is **FINISHED**, inspect and retrieve the output:

```
$ tapis jobs history 4e972f77-5bf9-446e-87a2-3541c4ea5745-007
+-------------------+----------------+-----------------------------------------------
↪-------------------------------+
```

(continues on next page)

(continued from previous page)

```
| status             | created         | description                                                     ␣
↪                                |
+--------------------+-----------------+--------------------------------------------------
↪-----------------------------+
| PENDING            | 8 minutes ago   | Job processing beginning                                        ␣
↪                                |
| PROCESSING_INPUTS | 8 minutes ago   | Identifying input files for staging                             ␣
↪                                |
| STAGING_INPUTS     | 8 minutes ago   | Transferring job input data to execution␣
↪system                                |
| STAGING_INPUTS     | 8 minutes ago   | Job input copy in progress: agave://tacc.work.
↪taccuser/public/SP1.fq to agav |
|                    |                 | e://tacc.stampede2.taccuser//scratch/05896/
↪taccuser/taccuser/job-4e972f77-5b |
|                    |                 | f9-446e-87a2-3541c4ea5745-007-taccuser-fastqc_
↪app-job-1589377205989/SP1.fq   |
| STAGED             | 8 minutes ago   | Job inputs staged to execution system                           ␣
↪                                |
| STAGING_JOB        | 8 minutes ago   | Staging runtime assets to execution system                      ␣
↪                                |
| STAGING_JOB        | 8 minutes ago   | Fetching application assets from                                ␣
↪                                |
|                    |                 | agave://tacc.work.taccuser/taccuser/apps/
↪fastqc_app-0.11.9                     |
| STAGING_JOB        | 8 minutes ago   | Staging runtime assets to agave://tacc.
↪stampede2.taccuser//scratch/05896/sd2 |
|                    |                 | e0004/taccuser/job-4e972f77-5bf9-446e-87a2-
↪3541c4ea5745-007-taccuser-fastqc_ |
|                    |                 | app-job-1589377205989                                           ␣
↪                                |
| SUBMITTING         | 8 minutes ago   | Submitting job to execution system                              ␣
↪                                |
| QUEUED             | 7 minutes ago   | Job queued to execution system queue                            ␣
↪                                |
| RUNNING            | 3 minutes ago   | Job running on execution system                                 ␣
↪                                |
| CLEANING_UP        | 29 seconds ago | Job completed execution                                         ␣
↪                                |
| FINISHED           | 29 seconds ago | Job completed successfully                                      ␣
↪                                |
+--------------------+-----------------+--------------------------------------------------
↪-----------------------------+

$ tapis jobs outputs list 4e972f77-5bf9-446e-87a2-3541c4ea5745-007
+-----------------------------------------------------------------------------------------
↪+------------------+--------+
| name                                                                                    ␣
↪| lastModified     | length |
+-----------------------------------------------------------------------------------------
↪+------------------+--------+
| SP1.fq                                                                                  ␣
↪| 21 minutes ago   |  22471 |
| SP1_fastqc.html                                                                         ␣
↪| 17 minutes ago   | 561767 |
| SP1_fastqc.zip                                                                          ␣
↪| 17 minutes ago   | 420233 |
| container_exec.log                                                                      ␣
↪| 17 minutes ago   |  19232 |
```

(continues on next page)

(continued from previous page)

```
| lib                                                                          ␣
↪| 17 minutes ago  |   4096 |
| runner.sh                                                                    ␣
↪| 17 minutes ago  |    875 |
| taccuser-fastqc_app-job-1589377205989-4e972f77-5bf9-446e-87a2-3541c4ea5745-007.err␣
↪| 21 minutes ago  |    372 |
| taccuser-fastqc_app-job-1589377205989-4e972f77-5bf9-446e-87a2-3541c4ea5745-007.out␣
↪| 21 minutes ago  |     29 |
| taccuser-fastqc_app-job-1589377205989.ipcexe                                 ␣
↪| 21 minutes ago  |   2772 |
| tester.sh                                                                    ␣
↪| 21 minutes ago  |     44 |
+----------------------------------------------------------------------------------
↪+-----------------+--------+

$ tapis jobs outputs download 4e972f77-5bf9-446e-87a2-3541c4ea5745-007
Walking remote resource...
Found 13 file(s) in 5s
Downloading .agave.archive...
Downloading .agave.log...
Downloading container_exec.log...
Downloading container_exec.sh...
Downloading VERSION...
Downloading runner.sh...
Downloading taccuser-fastqc_app-job-1589377205989-4e972f77-5bf9-446e-87a2-
↪3541c4ea5745-007.err...
Downloading taccuser-fastqc_app-job-1589377205989-4e972f77-5bf9-446e-87a2-
↪3541c4ea5745-007.out...
Downloading taccuser-fastqc_app-job-1589377205989.ipcexe...
Downloading SP1.fq...
Downloading SP1_fastqc.html...
Downloading SP1_fastqc.zip...
Downloading tester.sh...
Downloaded 13 files in 61s
+-------------+-------+
| Field       | Value |
+-------------+-------+
| downloaded  | 13    |
| skipped     | 0     |
| messages    | 0     |
| elapsed_sec | 66    |
+-------------+-------+
```

If the file `SP1_fastq.html` exists, then the run was successful.

12.2.4 Best Practices and Next Steps

Congratulations! You built and deployed your own Tapis. Consider the following recommended best practices when building app bundles, and share your work with others in the following ways:

Best Practices for Developing Containerized App Bundles

The app development approach demonstrated here is meant to be *flexible*, in that it can adapt to many different scientific applications and data sets, and *scalable*, in that it can run efficiently on TACC peta-scale systems. Some tips for the app developer in building new apps:

1. Write a clean Dockerfile so there is no question of source code / version provenance. Minimize image size where possible by removing, e.g. source code tarballs and installation directories.
2. Design a robust, but small and portable test case to package with the app bundle. Make liberal use of error checking in `tester.sh` and `runner.sh`.
3. Use only command line arguments when calling the containerized executable (with the `container_exec` function). If the executable requires a configuration file, use a wrapper script inside the container to parse inputs from the command line and generate the appropriate configuration file.
4. Explicitly declare all inputs, and explicitly write all outputs. This includes file name and full path.
5. Package and curate outputs into a user-friendly format. Some use cases may benefit from a tarball of all output files; some use cases may benefit from individual files.
6. Make output file names deterministic and predictable to facilitate scripting and job chaining.
7. Document all expected outputs in the `tester.sh` and `runner.sh` wrapper scripts. Where appropriate, validate output and provide helpful error messaging.
8. Share your Docker images and app bundles with the SD2E community to benefit others and elicit feedback.

Best practices were adapted from the Computational Genomics Lab.

Put Your App under Version Control

As you iterate with new changes in your codebase, or as new versions of your software are released, you will want to deploy updated copies of your app. Keeping the app bundle directory under version control (e.g. with Git) enables the developer to make incremental updates to the app version without losing any past information.

Many Tapis tenants maintain organizations in Github or Gitlab. Please consult with other members of your tenant and consider contributing your app bundle repo to the organization. Doing so promotes transparency into the function of an app, and enables others to build their own copies.

Considerations for Docker Images

Some Tapis tenants maintain Docker Hub organizations. Consult with your tenant admin to find out if it would be appropriate to push your image into the organization name space. Doing so may give added benefits in terms of privacy of sensitive code and provenance. In addition, always make sure to name your images with a descriptive name, an tag appropriate to the version, and **LABEL** your image with a maintainer.

CHAPTER 13

Modify an Existing App

Modifying an existing app would come in handy if there is a need to change the, e.g., execution system, description, max run time, default inputs, parameter descriptions, or a number of other things. As an example, the guide below goes through the process of adding a reference to default input data.

13.1 Modify the App Json

To modify a Tapis app after it has been deployed, edit the original app `json` file and use the command line interface to push the changes to the tenant. Below is an example app `json` file from a previous section of this how-to guide:

```
{
  "checkpointable": false,
  "name": "{{ username }}-{{ app.name }}",
  "executionSystem": "{{ app.execution_system }}",
  "executionType": "HPC",
  "deploymentPath": "{{ username }}/apps/{{ app.name }}-{{ app.version }}",
  "deploymentSystem": "{{ app.deployment_system }}",
  "helpURI": "",
  "label": "{{ app.label }}",
  "shortDescription": "{{ app.description }}",
  "longDescription": "",
  "modules": [
    "load tacc-singularity"
  ],
  "ontology": [],
  "parallelism": "SERIAL",
  "tags": [],
  "templatePath": "runner.sh",
  "testPath": "tester.sh",
  "version": "{{ app.version }}",
  "defaultMaxRunTime": "00:30:00",
  "inputs":[
    {
```

(continues on next page)

(continued from previous page)

```
      "id": "fastq",
      "value": {
        "default": "",
        "visible": true,
        "required": true
      },
      "semantics": {
        "ontology": [
          "http://edamontology.org/format_1930"
        ]
      },
      "details": {
        "label": "FASTQ sequence file"
      }
    }
  ],
  "parameters": [
    {
      "id": "CONTAINER_IMAGE",
      "value": {
        "default": "{{ docker.namespace }}/{{ docker.repo }}:{{ docker.tag }}",
        "type": "string",
        "visible": false,
        "required": true,
        "order": 1000
      }
    }
  ],
  "outputs": []
}
```

If you followed the Create a Custom App how-to guide, then you may have a sample fastq file located here:

```
agave://data-tacc-work-username/test-data/SP1.fq
```

Modify the original `app.json` file to include the complete URI to this sample data as follows:

```
{
"inputs":[
  {
    "id": "fastq",
    "value": {
      "default": "agave://tacc.work.taccuser/test-data/SP1.fq",
      "visible": true,
       "required": true
    }
 }
```

13.2 Update the App

Then, push the app update by performing the following:

```
$ tapis apps update -F app.json taccuser-fastqc_app-0.11.9
```

If successful, Tapis will automatically increment the revision number associated with the app. To confirm, use the `tapis apps show` command:

```
$ tapis apps show -c id -c revision taccuser-fastqc_app-0.11.9
+----------+----------------------------+
| Field    | Value                      |
+----------+----------------------------+
| id       | taccuser-fastqc_app-0.11.9 |
| revision | 2                          |
+----------+----------------------------+
```

13.3 Further Help

Additional fields that can be used in app descriptions can be found in the Tapis Documentation.

CHAPTER 14

Share an App with Others

As a standard user of Tapis tenants, you have permissions to build and deploy private apps only. Private apps are, by default, only visible to you. To share an app with your colleagues, follow the steps below.

14.1 Update Permissions on an App

Assuming you have a private app called `taccuser-fastqc_app-0.11.9` (developed earlier in this how-to guide), you can check who has permissions to access the app with the following command:

```
$ tapis apps pems list taccuser-fastqc_app-0.11.9
+----------+------+-------+---------+
| username | read | write | execute |
+----------+------+-------+---------+
| taccuser | True | True  | True    |
+----------+------+-------+---------+
```

By default, the creator of an app is the only one with read, write, or execute privileges. Next, identify the tenant username of the user with whom you would like to share the app.

Tip: See this page for an example on how to find another user's username

Given the username `jdoe`, grant that user permissions with:

```
$ tapis app pems grant taccuser-fastqc_app-0.11.9 jdoe ALL
+----------+------+-------+---------+
| username | read | write | execute |
+----------+------+-------+---------+
| taccuser | True | True  | True    |
| jdoe     | True | True  | True    |
+----------+------+-------+---------+
```

Ask your collaborator (`jdoe`) to perform the `tapis apps list` command, and they should now be able to see your app.

14.2 Update Permissions on an Execution System

Before your collaborator (`jdoe`) can run a job with your private app, they must also have correct permissions on the execution system associated with the app.

```
# Find the execution system associated with your private app
$ tapis apps show -c executionSystem taccuser-fastqc_app-0.11.9
+-----------------+-------------------------+
| Field           | Value                   |
+-----------------+-------------------------+
| executionSystem | tacc.stampede2.taccuser |
+-----------------+-------------------------+

# List permissions on the execution system
$ tapis systems roles list tacc.stampede2.taccuser
+----------+-------+
| username | role  |
+----------+-------+
| taccuser | OWNER |
+----------+-------+

# Grant 'USER' permission to your collaborator
$ tapis systems roles grant tacc.stampede2.taccuser jdoe USER
+----------+-------+
| username | role  |
+----------+-------+
| taccuser | OWNER |
| jdoe     | USER  |
+----------+-------+
```

Ask your collaborator to perform the `tapis systems list` command, and they should now be able to see your private system. Now, your collaborator can run jobs against your private app using the same `job.json` file and `tapis jobs submit` commands as you.

14.3 Publish the App Globally

Standard users do not have the appropriate permissions to make an app public, thereby sharing it with all users on the tenant. If you have deployed and tested an app, and think it would be of general use to the community, please contact your tenant admin to ask for information on publishing your app.

CHAPTER 15

Set up a Workflow

This guide will demonstrate a few common ways users can leverage the Tapis CLI for complex workflows.

15.1 Chain Multiple Apps

A useful feature of Tapis is the ability to use the output of one job as the input of a second job. It saves time and avoids unnecessary file transfers by keeping the input / output within Tapis "job API space". For example, consider the following two hypothetical apps:

App Name	Input	Output
ImageClassifier-1.0	image.jpg	*report.txt**
FileZipper-1.0	*report.txt**	report.txt.zip

In these examples, the first app, ImageClassifier-1.0 takes an image file as input, and produces an output report.txt file. The second app, FileZipper-1.0, takes the output report.txt and compresses it to report.txt.zip.

First, submit a job against the first app:

```
# Assemble job1.json file
$ cat job1.json
{
  "name": "Image Classifier Demo Job",
  "appId": "ImageClassifier-1.0",
  "archive": false,
  "inputs": {
    "image": "agave://tacc.work.taccuser/test-data/image.jpg"
  },
  "parameter": {}
}

# Submit the job
$ tapis jobs submit -F job1.json
```

(continues on next page)

(continued from previous page)

```
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | f0cb69a1-63a4-4970-9921-843968e66723-007  |
| name   | Image Classifier Demo Job                |
| status | ACCEPTED                                 |
+--------+------------------------------------------+

# When it is done, list the outputs
$ tapis jobs outputs list f0cb69a1-63a4-4970-9921-843968e66723-007
+------------+---------------+---------+
| name       | lastModified  |  length |
+------------+---------------+---------+
| ...        |               |         |
| report.txt | 2 minutes ago |  116625 |
| ...        |               |         |
+------------+---------------+---------+
```

Now that the first job is done, prepare and submit the second job referencing the output from the first job. The URI pointing to the report.txt file takes a very specific form:

```
# Assemble job2.json file
$ cat job2.json
{
  "name": "File Zipper Demo Job",
  "appId": "FileZipper-1.0",
  "archive": false,
  "inputs": {
    "file": "https://api.tacc.utexas.edu/jobs/v2/f0cb69a1-63a4-4970-9921-843968e66723-
↪007/outputs/media/report.txt"
  },
  "parameter": {}
}

# Submit the job
$ tapis jobs submit -F job2.json
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | 3fea4b88-424a-4c25-b0ef-6e6908eed843-007  |
| name   | File Zipper Demo Job                     |
| status | ACCEPTED                                 |
+--------+------------------------------------------+

# When it is done, list the outputs
$ tapis jobs outputs list 3fea4b88-424a-4c25-b0ef-6e6908eed843-007
+----------------+---------------+--------+
| name           | lastModified  | length |
+----------------+---------------+--------+
| ...            |               |        |
| report.txt.zip | 2 minutes ago |   2906 |
| ...            |               |        |
+----------------+---------------+--------+
```

If everything worked correctly, you should now see the final zipped file when listing the job outputs. As long as you remain on the **tacc.prod** tenant, much of the URI to the temporary file will remain the same:

```
https://api.tacc.utexas.edu/      # base URL - DO NOT CHANGE
jobs/v2/                          # refers to jobs API - DO NOT CHANGE
f0cb69a1-63a4-4970-9921-.../      # job ID from step 1
outputs/media/                    # location of output data - DO NOT CHANGE
report.txt                        # name of output file from step 1
```

Finally, the zipped report can be downloaded as:

```
$ tapis jobs outputs download 3fea4b88-424a-4c25-b0ef-6e6908eed843-007 report.txt.zip
+-------------+-------+
| Field       | Value |
+-------------+-------+
| downloaded  | 1     |
| skipped     | 0     |
| messages    | 0     |
| elapsed_sec | 8     |
+-------------+-------+
```

15.2 Parameter Sweeps

Many public apps are designed to take one input file or configuration, run an analysis, and return a result. With some simple scripting, it is possible to perform parameter sweeps using multiple Tapis jobs. For example, imagine you would like to run FastQC on a series of FASTQ files named: `fastq_01.fq`, `fastq_02.fq`, `fastq_03.fq`, etc.:

```
# Organize the data in a local folder:
$ ls fastq_data/
fastq_01.fq  fastq_02.fq  fastq_03.fq

# Upload all fastqc files to a storage system
$ tapis files upload agave://tacc.work.taccuser/test-data/ fastq_data
+-------------------+----------+
| Field             | Value    |
+-------------------+----------+
| uploaded          | 3        |
| skipped           | 0        |
| messages          | 0        |
| bytes_transferred | 65.83 kB |
| elapsed_sec       | 8        |
+-------------------+----------+

# Create a template json file:
$  tapis jobs init --no-archive --no-notify taccuser-fastqc_app-0.11.9 > job_template.
↪json
$ cat job_template.json
{
  "name": "taccuser-fastqc_app-job-1589474193147",
  "appId": "taccuser-fastqc_app-0.11.9",
  "archive": false,
  "inputs": {
    "fastq": "agave://tacc.work.taccuser/public/SP1.fq"
  },
  "parameters": {}
}
```

The next step is to write a short script around the `job_template.json` template that populates the file name into a new job json file, then submits the ob. Here is an example script:

```
#!/bin/bash

for FILE in ` ls fastq_data/ `
do

cat <<EOF >fastqc.json
{
  "name": "FastQC $FILE",
  "appId": "taccuser-fastqc_app-0.11.9",
  "archive": false,
  "inputs": {
    "fastq": "agave://tacc.work.taccuser/test-data/fastq_data/$FILE"
  },
  "parameters": {}
}
EOF

tapis jobs submit -F fastqc.json

done
```

Finally, execute the script:

```
$ bash parameter_sweep.sh
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | eef0030f-77d6-4a98-8893-a385137c3b44-007 |
| name   | FastQC fastq_01.fq                       |
| status | ACCEPTED                                 |
+--------+------------------------------------------+
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | 130a2099-3ae5-4f2c-ab2d-840a642cb2a9-007 |
| name   | FastQC fastq_02.fq                       |
| status | ACCEPTED                                 |
+--------+------------------------------------------+
+--------+------------------------------------------+
| Field  | Value                                    |
+--------+------------------------------------------+
| id     | 73ace158-7195-463f-b436-c4a519f7ba83-007 |
| name   | FastQC fastq_03.fq                       |
| status | ACCEPTED                                 |
+--------+------------------------------------------+

$ tapis jobs list --limit 3
+------------------------------------------+--------------------+--------+
| id                                       | name               | status |
+------------------------------------------+--------------------+--------+
| 73ace158-7195-463f-b436-c4a519f7ba83-007 | FastQC fastq_03.fq | QUEUED |
| 130a2099-3ae5-4f2c-ab2d-840a642cb2a9-007 | FastQC fastq_02.fq | QUEUED |
| eef0030f-77d6-4a98-8893-a385137c3b44-007 | FastQC fastq_01.fq | QUEUED |
+------------------------------------------+--------------------+--------+
```

This is a simple example of a control script with plenty of room for advanced features and error checking.

CHAPTER 16

Work with Actors

In Tapis, **actors** are container-based functions-as-a-service that follow the actor model of concurrent computation. An actor responds to messages it receives by changing its state, performing an action, sending out response messages, or all of the above.

The function an actor performs is exposed as the default command in a container. It is typically quick and requires little processing power - i.e. an app may be configured to run FastQC, and an actor may trigger a job using that app.

The guide below is a brief introduction to interacting with actors on the Tapis platform. For a full reference guide to actors, see the Abaco Documentation.

16.1 Create a New Actor

The function of an actor is exposed as the default command in a Docker container. Here, we will create an actor from an existing Docker container image called **tacc/hello-world:latest** available on Docker Hub. The default command for this container simply prints the message "Hello, World" or the message sent to it, which will be captured in the actor logs.

Create the actor as:

```
$ tapis actors create --repo tacc/hello-world:latest \
                      -n example-actor \
                      -d "Test actor that says Hello, World"
+----------------+----------------------------+
| Field          | Value                      |
+----------------+----------------------------+
| id             | NN5N0kGDvZQpA              |
| name           | example-actor              |
| owner          | taccuser                   |
| image          | tacc/hello-world:latest    |
| lastUpdateTime | 2021-07-14T22:25:06.171534 |
| status         | SUBMITTED                  |
| cronOn         | False                      |
+----------------+----------------------------+
```

The `--repo` flag points to the Docker Hub repo on which this actor is based, the `-n` flag and `-d` flag attach a human-readable name and description to the actor, the `-e` flags demonstrate how to set (optional) environment variables for the actor.

The resulting actor is assigned an id: `NN5N0kGDvZQpA`. The actor id can be queried by:

```
$ tapis actors show -v NN5N0kGDvZQpA
{
 "id": "NN5N0kGDvZQpA",
 "name": "example-actor",
 "description": "Test actor that says Hello, World",
 "owner": "sgopal",
 "image": "tacc/hello-world:latest",
 "createTime": "2021-07-14T22:25:06.171Z",
 "lastUpdateTime": "2021-07-14T22:25:06.171Z",
 "defaultEnvironment": {},
 "gid": 862347,
 "hints": [],
 "link": "",
 "mounts": [],
 "privileged": false,
 "queue": "default",
 "stateless": true,
 "status": "READY",
 "statusMessage": " ",
 "token": true,
 "uid": 862347,
 "useContainerUid": false,
 "webhook": "",
 "cronOn": false,
 "cronSchedule": null,
 "cronNextEx": null,
 "_links": {
   "executions": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA/executions",
   "owner": "https://api.tacc.utexas.edu/profiles/v2/sgopal",
   "self": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA"
   }
 }
```

Above, you can see the plain text name, description, and any default environment variables that were passed on the command line. In addition, you can see the "status" of the actor is "READY", meaning it is ready to receive and act on messages. Finally, you can list all actors visible to you with:

```
$ tapis actors list
+---------------+---------------+----------+------------------------------+------------
↪----------------+--------+-------+
| id            | name          | owner    | image                        |␣
↪lastUpdateTime               | status | cronOn|
+---------------+---------------+----------+------------------------------+------------
↪----------------+--------+-------+
| NN5N0kGDvZQpA | example-actor | taccuser | tacc/hello-world:latest      | 2021-07-
↪14T22:25:06.171Z    | READY  | False |
+---------------+---------------+----------+------------------------------+------------
↪----------------+--------+-------+
```

16.2 Probe the Underlying Container

An actor now exists and is waiting for a message to respond to. But, how will the actor respond when sent a message? We can probe the underlying container to figure out what this specific actor will do. First pull the container locally:

```
$ docker pull tacc/hello-world:latest
latest: Pulling from tacc/hello-world
Digest: sha256:baf7241b9d6fb1b123825021b831337307b9fa0aa4d45b14c9405ebf2a36a929
Status: Image is up to date for tacc/hello-world:latest
docker.io/tacc/hello-world:latest
```

Then find the default command for the container:

```
$ docker inspect tacc/hello-world:latest | jq ".[].ContainerConfig.Cmd"
[
 "/bin/sh",
 "-c",
 "#(nop) ",
 "CMD [\"python\" \"/hello_world.py\"]"
]
```

It runs `hello_world.py` at the root level. Print out the contents of `hello_world.py` to inspect:

```
$ docker run --rm tacc/hello-world:latest cat /hello_world.py
```

```
1   """Say Hello, World or the message received from user input"""
2   from agavepy.actors import get_context
3
4   def say_hello_world(m):
5   """Print message from user if present, else echo "Hello, World"""
6       if m == " ":
7           print("Actor says: Hello, World")
8       else:
9           print("Actor received message: {}".format(m))
10
11  def main():
12  """Main entry to grab message context from user input"""
13      context = get_context()
14      message = context['raw_message']
15      say_hello_world(message)
16
17  if __name__ == '__main__':
18      main()
```

This container, when run, will first get the message that was passed to it (from the `get_context()` function, line 10). Then it will print various parts of the message and the environment.

16.3 Submit a Message to the Actor

Next, let's craft a simple message to send to the reactor. Messages can be plain text or in JSON format. When using the python actor libraries as in the example above, JSON-formatted messages are made available as python dictionaries.

```
# Write a message
$ export MESSAGE='Hello, World'
```

(continues on next page)

(continued from previous page)

```
$ echo $MESSAGE
Hello, World

$ Submit the message to the actor
$ tapis actors submit -m "$MESSAGE" NN5N0kGDvZQpA
+-------------+---------------+
|  Field      | Value         |
+-------------+---------------+
| executionId | N4xQ5WM5Np1X0 |
| msg         | Hello, World  |
+-------------+---------------+
```

The id of the actor (`N4xQ5WM5Np1X0`) was used on the command line to specify which actor should receive the message. In response, an "execution id" (`N4xQ5WM5Np1X0`) is returned. An execution is a specific instance of an actor. List all the executions for a given actor as:

The above execution has already completed. Show detailed information for the execution with:

```
$ tapis actors execs show -v boEg3mEvrKO5w ayB45Oe8GJvAA
{
   "actorId": "NN5N0kGDvZQpA",
   "apiServer": "https://api.tacc.utexas.edu",
   "cpu": 121748743,
   "exitCode": 0,
   "finalState": {
     "Dead": false,
     "Error": "",
     "ExitCode": 0,
     "FinishedAt": "2021-07-14T22:32:45.602Z",
     "OOMKilled": false,
     "Paused": false,
     "Pid": 0,
     "Restarting": false,
     "Running": false,
     "StartedAt": "2021-07-14T22:32:45.223Z",
     "Status": "exited"
   },
   "id": "N4xQ5WM5Np1X0",
   "io": 176,
   "messageReceivedTime": "2021-07-14T22:32:37.051Z",
   "runtime": 1,
   "startTime": "2021-07-14T22:32:44.752Z",
   "status": "COMPLETE",
   "workerId": "JABKl4BeDwXJD",
   "_links": {
     "logs": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA/executions/
↪N4xQ5WM5Np1X0/logs",
     "owner": "https://api.tacc.utexas.edu/profiles/v2/sgopal",
     "self": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA/executions/
↪N4xQ5WM5Np1X0"
   }
}
```

16.4 Check the Logs for an Execution

An execution's logs will contain whatever was printed to STDOUT / STDERR by the actor. In our demo actor, we just expect the actor to print the message passed to it.

```
$ tapis actors execs logs NN5N0kGDvZQpA N4xQ5WM5Np1X0
Logs for execution N4xQ5WM5Np1X0
 Actor received message: Hello, World
```

Sure enough, the information in the execution logs match what we expected `hello_world.py` to print. The message was pulled in by the `get_context()` function. It was not done in this script, but in a normal scenario, the actor would then act on the contents of that message to, e.g., kick off a job, perform some data management, send messages to other actors, or more.

16.5 Run Synchronously

The previous message submission (with `tapis actors submit`) was an *asynchronous* run, meaning the command prompt detached from the process after it was submitted to the actor. In that case, it was up to us to check the execution to see if it had completed and manually print the logs.

There is also a mode to run actors *synchronously* using `tapis actors run`, meaning the command line stays attached to the process awaiting a response after sending a message to the actor. For example:

```
$ tapis actors run -m "$MESSAGE" NN5N0kGDvZQpA
FULL CONTEXT:
{
  "username": "taccuser",
  "HOSTNAME": "33d4dd334ef9",
  "_abaco_worker_id": "X5xGkZ0lol0D3",
  "raw_message": "Hello, World",
  "actor_dbid": "TACC-PROD_boEg3mEvrKO5w",
  "new_foo": "new_bar",
  "_abaco_container_repo": "tacc/hello-world:latest",
  "content_type": null,
  "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
  "MSG": "{\"key1\":\"value1\", \"key2\":\"value2\"}",
  "HOME": "/",
  "_abaco_actor_state": "{}",
  "_abaco_actor_name": "example-actor",
  "_abaco_Content_Type": "str",
  "execution_id": "jP3RExQW108wM",
  "_abaco_synchronous": "True",
  "_abaco_access_token": "de6d11bdbb5a16bdd85beec692b1b283",
  "message_dict": {
    "key2": "value2",
    "key1": "value1"
  },
  "_abaco_api_server": "https://api.tacc.utexas.edu",
  "_abaco_actor_dbid": "TACC-PROD_boEg3mEvrKO5w",
  "_abaco_jwt_header_name": "X-Jwt-Assertion-Tacc-Prod",
  "_abaco_actor_id": "boEg3mEvrKO5w",
  "_abaco_execution_id": "jP3RExQW108wM",
  "state": "{}",
  "_abaco_username": "taccuser",
  "actor_id": "boEg3mEvrKO5w"
```

(continues on next page)

(continued from previous page)

```
}
...
```

The output above is truncated because it is mostly the same response as our first execution of the actor. This time, however, we did not need to query the logs for this execution for them to print to screen - that was done automatically. In addition, the new environment variable settings can be seen in the context (see highlighted line).

16.6 Delete an Actor

Similar to other resources in Tapis, actors can be deleted with the following:

```
$ tapis actors delete NN5N0kGDvZQpA
+----------+-------------------+
| Field    | Value             |
+----------+-------------------+
| deleted  | ['NN5N0kGDvZQpA'] |
| messages | []                |
+----------+-------------------+
```

This will delete the actor and any associated executions.

CHAPTER 17

Initialize a new Tapis Actor project

This guide will demonstrate how to create a custom actor from scratch. It is assumed you are already familiar with how to Work with Actors. In this example, we will build a simple word count actor that counts and prints the number of words in a provided message.

We will demonstrate how to initialize an actor project from scratch.

17.1 Create a project "hello_world_actor"

To get started with creating an actor, running the `tapis actors init` command will fetch a very simple code skeleton you can fill in and deploy.

For example:

```
$ tapis actors init

+-------+----------------------------------------------------+
| stage | message                                            |
+-------+----------------------------------------------------+
| setup | Project path: ./new_actor                          |
| setup | CookieCutter variable name=new_actor               |
| setup | CookieCutter variable project_slug=new_actor       |
| setup | CookieCutter variable docker_namespace=reshg       |
| setup | CookieCutter variable docker_registry=e            |
| clone | Project path: ./new_actor                          |
+-------+----------------------------------------------------+
```

Note: There are many project templates you can start working with. See tapis actors init –list-templates for an up to date listing.

```
$ tapis actors init --list-templates
+---------------------+---------------------+------------------------------------------
↪-------------+----------+
| id                  | name                | description                               ␣
↪             | level    |
+---------------------+---------------------+------------------------------------------
↪-------------+----------+
| default             | Default             | Basic code and configuration skeleton     ␣
↪             | beginner |
| echo                | Echo                | Echo message                              ␣
↪             | beginner |
| hello_world         | Hello World         | Say Hello, World!                         ␣
↪             | beginner |
| sd2e_base           | sd2e_base           | Default reactor context for               ␣
↪             | beginner |
|                     |                     | docker://sd2e/reactors:python3            ␣
↪             |          |
| tacc_reactors_base  | tacc_reactors_base  | Default actor context for                 ␣
↪             | beginner |
|                     |                     | docker://sd2e/reactors:python3            ␣
↪             |          |
+---------------------+---------------------+------------------------------------------
↪-------------+----------+
```

To use one of these templates:

```
$ tapis actors init --template hello_world
```

17.2 Components of an Actor

The new_actor/ project would contain the following files:

```
$ tree ../new_actor/
new-actor/
├── Dockerfile
├── project.ini
├── config.yml
├── default.py
├── requirements.txt
├── secrets.jsonsample
└── message.jsonschema
```

17.3 Write the Actor Function

The `default.py` script can be renamed to `hello_world.py`. The python script is where the code for your main function can be found. An example of a functional actor that says "Hello, World" is:

```
"""Say Hello, World or the message received from user input"""
from agavepy.actors import get_context

# function to print the message
def say_hello_world(m):
```

(continues on next page)

(continued from previous page)

```
"""Print message from user if present, else echo "Hello, World"""
    if m == " ":
        print("Actor says: Hello, World")
    else:
        print("Actor received message: {}".format(m))

def main():
"""Main entry to grab message context from user input"""
    context = get_context()
    message = context['raw_message']
    say_hello_world(message)

if __name__ == '__main__':
    main()
```

This code makes use of the **agavepy** python library which we will install in the Docker container. The library includes an "actors" object which is useful to grab the message and other context from the environment. And, it can be used to interact with other parts of the Tapis platform. Add the above code to your `hello_world.py` file.

17.4 Define Requirements

The `requirements.txt` file may contain the dependencies required for a project. The default `requirements.txt` contains agavepy python package.

17.5 Create a Dockerfile

The only requirements are python and the agavepy python library, which is available through PyPi. These are mentioned in the `requirements.txt` file A bare-bones Dockerfile needs to satisfy those dependencies, add the actor python script, and set a default command to run the actor python script. The following lines should be present in your `Dockerfile`:

```
# pull base image
FROM python:3.7-alpine

# add requirements.txt to docker container
ADD requirements.txt /requirements.txt

# install requirements.txt
RUN pip3 install -r /requirements.txt

# add the python script to docker container
ADD hello_world.py /hello_world.py

# command to run the python script
CMD ["python", "/hello_world.py"]
```

Tip: Creating small Docker images is important for maintaining actor speed and efficiency

17.6 Runtime Preparation

1. Define secrets.json: Rename secrets.json.sample to secrets.json, and obtain the required values from the Infrastructure team for secrets.json.
2. Define message.jsonschema: Define the Schema for Actor launch message.

17.7 Build and Push the Dockerfile

The Docker image must be pushed to a public repository in order for the actor to use it. Use the following Docker commands in your local actor folder to build and push to a repository that you have access to:

```
# Build and tag the image
$ docker build -t taccuser/hello-world:1.0 .
Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM python:3.7-slim
...
Successfully built b0a76425e8b3
Successfully tagged taccuser/hello-world:1.0

# Push the tagged image to Docker Hub
$ docker push taccuser/hello-world:1.0
The push refers to repository [docker.io/taccuser/word-count]
...
1.0: digest: sha256:67cc6f6f00589d9ae83b99d779e4893a25e103d07e4f660c14d9a0ee06a9ddaf␣
↪size: 1995
```

17.8 Create the Actor

Next, create an actor referring to the Docker repository above.

```
$ tapis actors create --repo taccuser/hello-world:1.0 \
                      -n hello-world \
                      -d "Actor to say Hello, World"
+-----------------+-----------------------------+
| Field           | Value                       |
+-----------------+-----------------------------+
| id              | NN5N0kGDvZQpA               |
| name            | hello-world                 |
| owner           | taccuser                    |
| image           | taccuser/hello-world:1.0    |
| lastUpdateTime  | 2021-07-14T22:25:06.171534 |
| status          | SUBMITTED                   |
| cronOn          | False                       |
+-----------------+------------------------------+
```

After a few seconds, the actor should be in state "READY", meaning it is ready to accept and process messages. Verbosely show the actor metadata to see that it's status is "READY", it is pointing to the correct docker image, and that it received the environment variables from `environment.json`:

```
$ tapis actors show -v NN5N0kGDvZQpA
{
```

(continues on next page)

(continued from previous page)

```
"id": "NN5N0kGDvZQpA",
"name": "example-actor",
"description": "Test actor that says Hello, World",
"owner": "sgopal",
"image": "tacc/hello-world:latest",
"createTime": "2021-07-14T22:25:06.171Z",
"lastUpdateTime": "2021-07-14T22:25:06.171Z",
"defaultEnvironment": {},
"gid": 862347,
"hints": [],
"link": "",
"mounts": [],
"privileged": false,
"queue": "default",
"stateless": true,
"status": "READY",
"statusMessage": " ",
"token": true,
"uid": 862347,
"useContainerUid": false,
"webhook": "",
"cronOn": false,
"cronSchedule": null,
"cronNextEx": null,
"_links": {
  "executions": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA/executions",
  "owner": "https://api.tacc.utexas.edu/profiles/v2/sgopal",
  "self": "https://api.tacc.utexas.edu/actors/v2/NN5N0kGDvZQpA"
  }
}
```

17.9 Run a Test Execution

Finally, pass a message to the actor to run a test execution. The number of words in the message should be returned in the actor execution logs:

```
# Send a message to the word-count actor
$ tapis actors submit -m "Hello, World" NN5N0kGDvZQpA
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| executionId | NN5N0kGDvZQpA                        |
| msg         | Hello, World                         |
+-------------+--------------------------------------+

# List executions of the word-count actor
$ tapis actors execs list NN5N0kGDvZQpA
+---------------+----------+
| executionId   | status   |
+---------------+----------+
| N4xQ5WM5Np1X0 | COMPLETE |
+---------------+----------+

# Get the logs from the completed actor execution
```

(continues on next page)

(continued from previous page)

```
$ tapis actors execs logs NN5N0kGDvZQpA N4xQ5WM5Np1X0
Logs for execution N4xQ5WM5Np1X0
 Actor received message: Hello, World
```

The actor can also be run synchronously using `tapis actors run`:

```
$ tapis actors run -m "Hello, World" NN5N0kGDvZQpA
Actor received message: Hello, World
```

17.10 Next Steps

Remember to put your actor under version control. Use a `.gitignore` file to avoid accidentally committing anything that contains API keys or passwords.

Please refer to the Abaco Documentation for more information on creating and working with actors.

CHAPTER 18

Find Additional Help

Please visit the Reference Docs for the Tapis CLI for additional support