SGCI Webinar: Portable, Scalable Computation with Containers and Abaco Functions

Joe Stubbs, PhD
Lead, Cloud and Interactive Computing

Texas Advanced Computing Center
University of Texas, Austin

TACC

About Me

- Educational background in Mathematics and theoretical Computer Science.
- Research Associate at Texas Advanced Computing Center for about 6.5 years.
- Formed the Cloud and Interactive Computing (CIC) group at TACC in March, 2017 (3 people).
- CIC focuses on cloud systems for research computing.
- Today CIC has 15 full time staff plus REU students and professional interns.
- Our work primarily funded by NSF but increasingly other agencies, including DARPA, CDC and NIH.

Webinar Outline

- Set the stage - an image classifier program in Python.
- Crash course introduction to containers and Docker.
- Introduce the serverless/Functions-as-a-Service (FaaS) model.
- Cover the basics of Abaco (Actor Based Containers) Platform.
- Walk through how to package our classifier program into a Docker image and register it as an Abaco function.
- Execute our classifier on the Abaco cloud.

An Image Classifier Program in Python

Assume we have a Python program that can classify an image

Utilizes Python, tensorflow and the requests library

Usage:
$ python classify_image.py --image_file <URL_to_image>

An Image Classifier Program in Python

```
$ python classify_image.py --image_file https://bit.ly/2WYkdby
```

An Image Classifier Program in Python

https://bit.ly/2WYkdby

An Image Classifier Program in Python

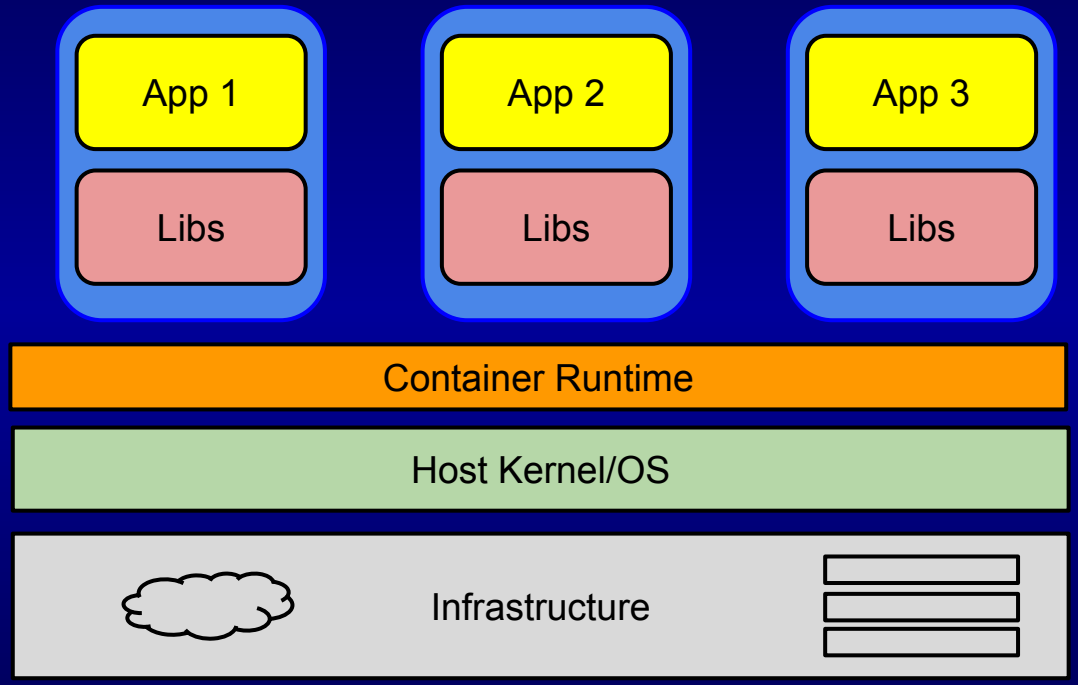
```
$ python classify_image.py --image_file https://bit.ly/2WYkdby

. . . .
Successfully downloaded inception-2015-12-05.tgz 88931400 bytes.
Labrador retriever (score = 0.97471)
golden retriever (score = 0.00324)
kuvasz (score = 0.00099)
bull mastiff (score = 0.00095)
Saint Bernard, St Bernard (score = 0.00067)
```

Containers: Reproducible Environments

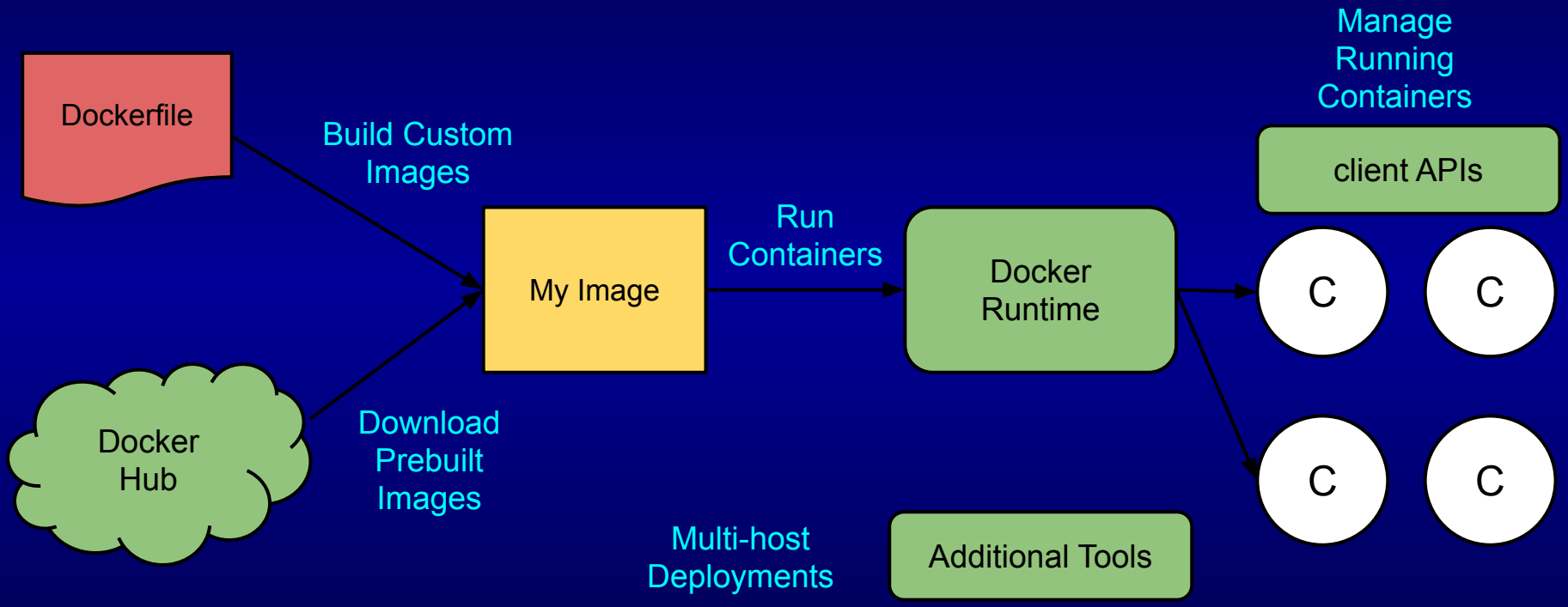
Isolated Userland Processes

Virtualized:
Network
I/O
CPU and MEM

Containers:
- Include all dependencies
- Ease installation
- Start up in miliseconds



Docker - A Container Platform



Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

- Text file with instructions for building a Docker image.
- Small set of reserved words, “FROM”, “RUN”, “ADD”, “CMD”, etc.
- Only the resulting changes to the file system matter.

Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

1) FROM instruction starts image with a pre-existing image.

Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

1) FROM instruction starts image with a pre-existing image.

2) RUN instruction runs arbitrary commands.

Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

1) FROM instruction starts image with a pre-existing image.

2) RUN instruction runs arbitrary commands.

3) ADD instruction adds local files to the image.

Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

1) FROM instruction starts image with a pre-existing image.

2) RUN instruction runs arbitrary commands.

3) ADD instruction adds local files to the image.

4) CMD instruction adds a default command to run in containers started from the resulting image.

Dockerfile

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

ADD classify_image.py /classify_image.py

CMD ["python3", "/classify_image.py"]
```

1) FROM instruction starts image with a pre-existing image.

2) RUN instruction runs arbitrary commands.

3) ADD instruction adds local files to the image.

4) CMD instruction adds a default command to run in containers started from the resulting image.

$ docker build -t jstubbs/sgci-classifier .

Running Docker Containers

- Given an image, run one or more containers from it with:

$ docker run <options> <image> <command>

- Options include mounting directories from the host and exposing ports.
- Each container started from a given image has a copy of the entire image file system.
- Any changes made to the file system by the running container do not impact the image or other running containers.

FaaS aka “Serverless”

What It Is

- Cloud computing model & software architecture.
- Roots in projectes like PiCloud, circa 2011.
- Started in 2014 (AWS Lambda).

How It works

- Register small programs (functions) to run on the cloud.
- Invoke the function through an API.
- Cloud provider manages the computing infrastructure where functions run.
- Enables applications to be developed without worrying about servers.

FaaS Pros and Cons

Pros

- Encourages modularity
- Independent scalability of components
- Automated scalability (let someone else manage the servers)

Cons

- Harder to predict performance
- Harder to reason about and debug

FaaS Platforms

Commercial

- AWS Lambda
- Google Cloud Functions
- MicroSoft Azure Functions
- GitLab Serverless
- IBM Cloud Functions

Open Source

- OpenFaas
- OpenWhisk
- IronFunctions
- SpringCloud Functions
- Kubeless

FaaS Platforms

Commercial

- AWS Lambda
- Google Cloud Functions
- MicroSoft Azure Functions
- GitLab Serverless
- IBM Cloud Functions

Open Source

- OpenFaas
- OpenWhisk
- IronFunctions
- SpringCloud Functions
- Kubeless

Abaco?

*Abaco - Introduction

Docker + Actor Model = Functional Computing Platform

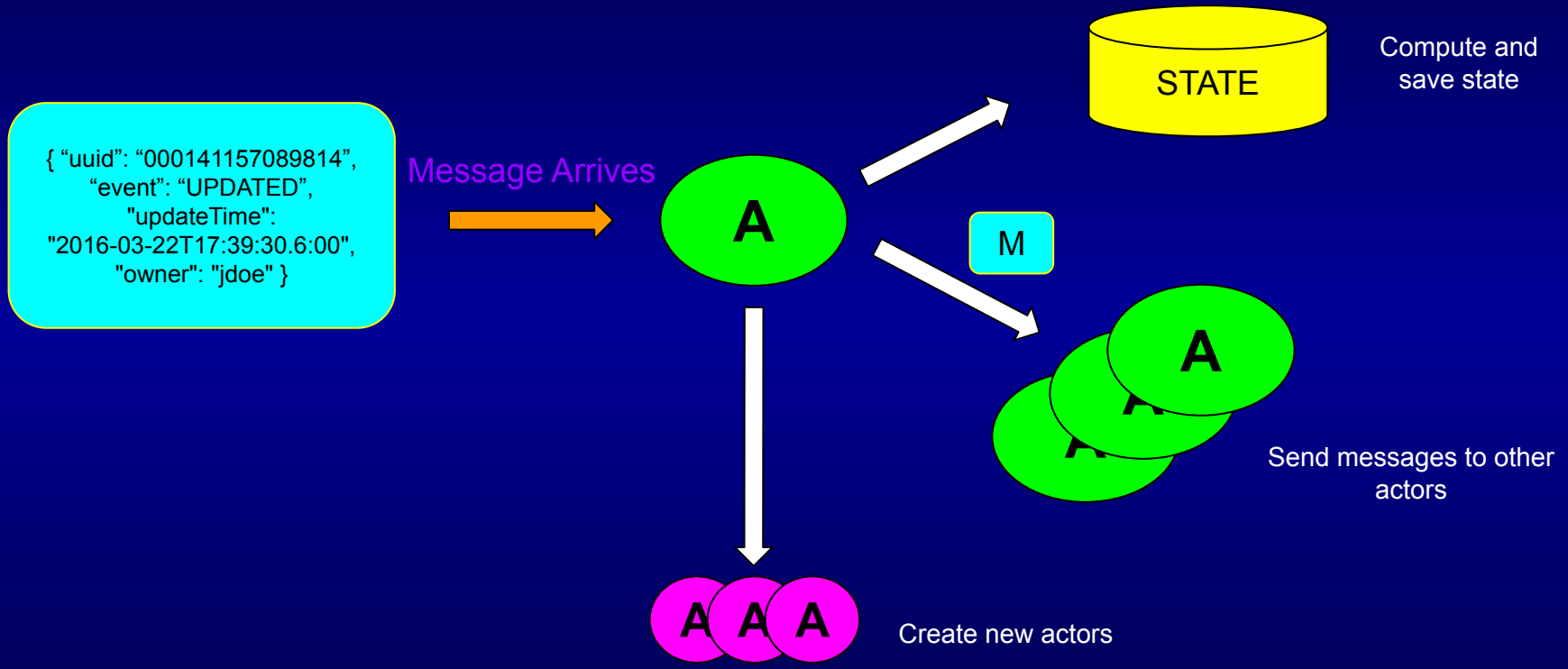
- “Severless” - users only interact with API
- Focus on research computing use cases, not enterprise services

Three Primary Capabilities

- “Reactors” for event-driven programming
- “Asynchronous Executors” for parallel function executions
- “Data Adapters” for building data services from disparate sources of data

* Work supported by grant #1740288 from the US National Science Foundation.

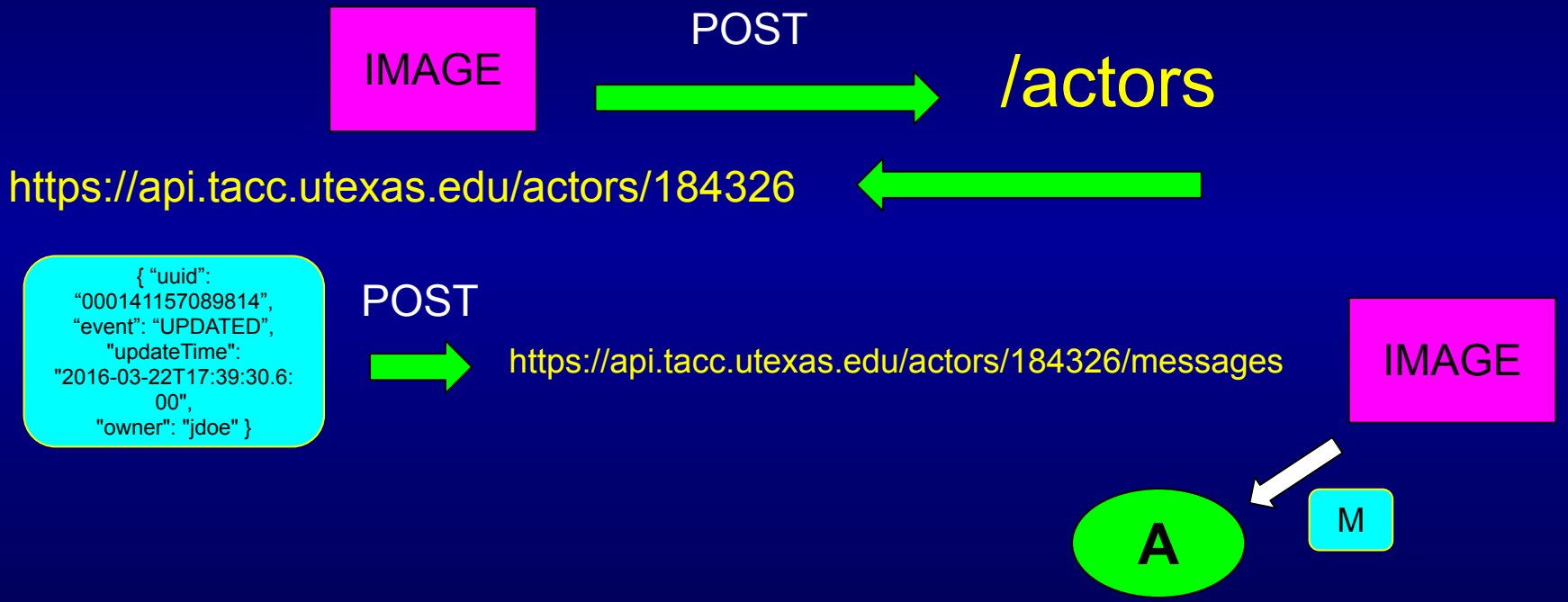
Actor Model


{ “uuid”: “000141157089814”,
“event”: “UPDATED”,
"updateTime":
"2016-03-22T17:39:30.6:00",
"owner": "jdoe" }
Message Arrives
A
STATE
Compute and save state
M
A
A
A
Send messages to other actors
A A A
Create new actors

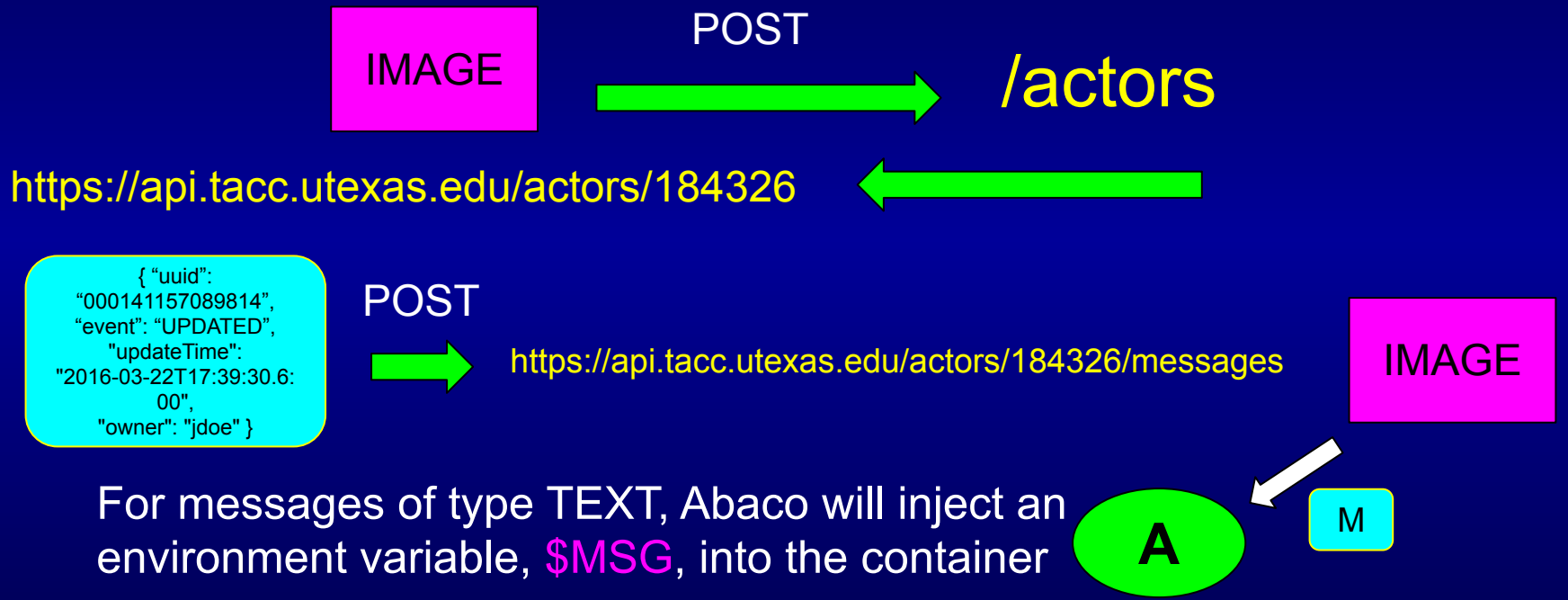
User-Defined Actors Via Docker

- Associate an actor with a Docker image.
- Assign the actor's inbox to a unique URI.
- Launch a container from the image in response to a message.

Abaco: Actor Based Containers



Abaco: Actor Based Containers



For messages of type TEXT, Abaco will inject an environment variable, $MSG, into the container

Abaco Compared To Other Platforms

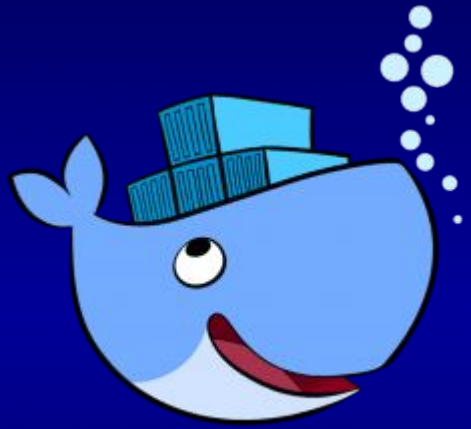
- Abaco is an open source project, funded by NSF, hosted at TACC and free to use for researchers
- Abaco leverages the Actor Model - state, aliases, links, etc.
- Abaco targets the research computing use case
  - Single container executions can run for hours
  - Access to more CPU cores, memory, even GPUs
- Abaco integrates with other TACC resources
  - Authentication for accessing other cloud APIs
  - POSIX interfaces to high performance TACC storage
  - TACC Jupyter environments for scaling notebook functions
- Abaco components can run at other institutions

Usage and Adoption

Usage since Jan, 2018

- 40,000 actors registered
- 600,000 executions
- 7M seconds of runtime
- 1.3*10^18 Jiffies CPU utilized

An Image Classifier Program on Abaco



ABACO

- Assume we have a Python script that can classify an image
- Works like: python classify_image.py --image_file <URL_to_image>
- We'll put it in Docker and run it on the Abaco cloud

Preparing the Image Classifier Program for Abaco

1. Create a Dockerfile for our program
2. Parse the $MSG environment variable Abaco will send
3. Set permissions in the image so that a non-root user can run our program.

```
FROM tensorflow/tensorflow:1.5.0-py3

RUN pip install requests

RUN mkdir /app
ADD classify_image.py /app/classify_image.py
ADD abaco.sh /app/abaco.sh
RUN chmod -R 777 /app

CMD ["/app/abaco.sh"]
```

https://github.com/joestubbs/faas-abaco-sgci-webinar

Preparing the Image Classifier Program, II

- We chose to write a small BASH script to integrate with Abaco.
- Separates original app from Abaco integration.
- All this does is execute out app and set the parameter, image_file, equal to $MSG

```
#!/bin/bash
# abaco.sh -- Entrypoint for Abaco-ready app

cd /app
python classify_image.py --image_file=$MSG
```

https://github.com/joestubbs/faas-abaco-sgci-webinar

Preparing the Image Classifier Program, III

- Build the image:

  $ docker build -t jstubbs/sgci-classifier .

- Test the image locally:

  $ docker run -e MSG=<some_URL> jstubbs/sgci-classifier

- Push image to Docker Hub:

  $ docker push jstubbs/sgci-classifier

*Registering the Classifier Actor

$ curl -H $TOKEN -d “image=jstubbs/sgci-classifier” https://api.tacc.utexas.edu/actors/v2


```
{
        'createTime': '2019-09-03 22:41:29.563024',
        'defaultEnvironment': {},
        'description': '',
        'id': 'O08Nzb3mRA7Bz',
        'image': 'jstubbs/sgci-classifier',
        'lastUpdateTime': '2019-09-03 22:41:29.563024',
        'mounts': [],
        'hints': [],
        'link': "",
        'name': '',
        'owner': 'jstubbs',
        'privileged': False,
        'state': {},
        'stateless': False,
        'status': 'SUBMITTED',
        'statusMessage': '',
        'type': 'none',
        'useContainerUid': False,
        'webhook': ""
}
```


- The id of the actor - needed for the next step.
- The status of the actor. Will move to READY once worker started and image pulled.

*Requires a TACC account and OAuth client registration

TACC

Executing Classifier Actor

```
$ curl -H $TOKEN -X POST -d “message=https://bit.ly/2WYkdby"
https://api.tacc.utexas.edu/actors/v2/O08Nzb3mRA7Bz/messages
{
    "executionId": "RrGp0wkEbJplo",
    "msg": "https://bit.ly/2WYkdby"
}
```

- Abaco responds immediately; the execution is asynchronous.
- The executionId is used to track the execution and retrieve results when it completes.

Executing Classifier Actor, II

$ curl -H $TOKEN
https://api.tacc.utexas.edu/actors/v2/O08Nzb3mRA7Bz/executions/RrGp0wkEbJplo


```
{
   "actorId": "YygyQkoZ65X0e",
   "cpu": 29441360370,
   "executor": "jstubbs",
   "exitCode": 0,
   "finalState": {
     "Error": "",
     "ExitCode": 0,
     "FinishedAt": "2019-09-07T20:00:14.629269037Z",
     "StartedAt": "2019-09-07T20:00:06.974068805Z",
   },
   "id": "RrGp0wkEbJplo",
   "io": 522702606,
   "messageReceivedTime": "2019-09-07 19:59:59.488538",
   "runtime": 9,
   "startTime": "2019-09-07 20:00:06.291694",
   "status": "COMPLETE",
}
```


- Abaco tracks various information abou thte execution, including resources utilized and metadata about the final state.
- When the execution completes, it's status will change to COMPLETE.

Retrieving Execution Logs

$ curl -H $TOKEN
https://api.tacc.utexas.edu/actors/v2/O08Nzb3mRA7Bz/executions/RrGp0wkEbJplo/logs

. . . .
Successfully downloaded inception-2015-12-05.tgz 88931400 bytes.
Labrador retriever (score = 0.97471)
golden retriever (score = 0.00324)
kuvasz (score = 0.00099)
bull mastiff (score = 0.00095)
Saint Bernard, St Bernard (score = 0.00067)

Alternative Clients

Abaco CLI

```
# create an actor:
$ abaco create jstubbs/sgci-classifier
# execute the actor
$ abaco submit -m <URL> O08Nzb3mRA7Bz
```

Abaco Python SDK

```
>>> cl.actors.list()
>>> cl.actors.add(body={“image”: “jstubbs/sgci-classifier”})
>>> cl.actors.sendMessage(actorId=O08Nzb3mRA7Bz, message=<URL>)
```

Next Steps

- Send your actor messages to classify thousands of images
- Give your actor a meaningful alias
- Share your actor/alias with other researchers
- Expose your actor in a web application/science gateway
- Explore additional Abaco features: https://abaco.readthedocs.io

Thanks!

Questions?

Docs: https://abaco.readthedocs.io

Github: https://github.com/TACC/abaco

Slack Team: https://tacc-cloud.slack.com, @cicsupport

Email: CICsupport@tacc.utexas.edu