ColonyOS Tutorial

⚙ Prerequisites

prerequisites

20 min	filename

Installation

You need to download and install two different CLI binaries.

- The Colonies CLI - Download here!
- The Pollinator CLI - Download here!

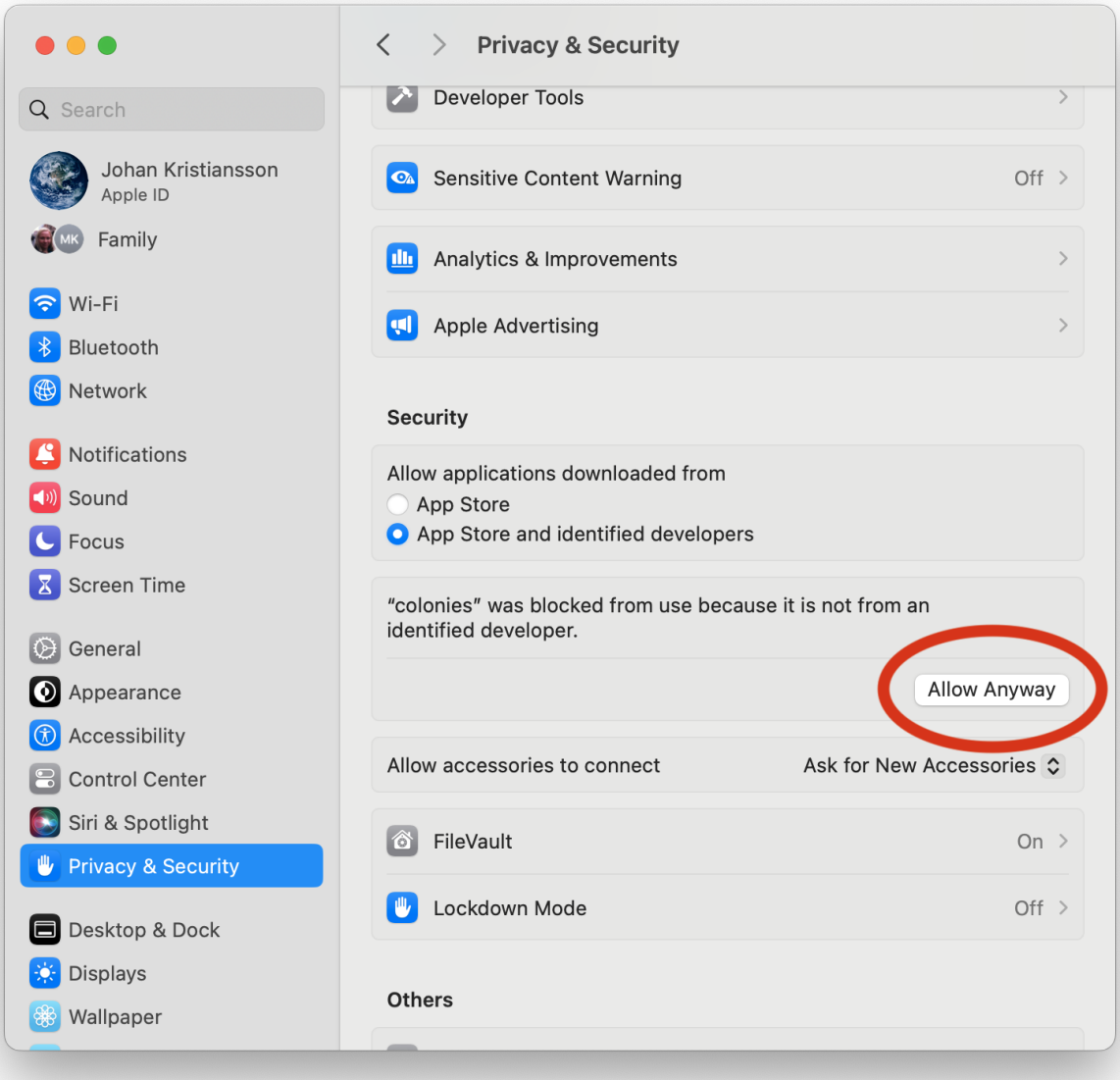
For Apple silicon, choose arm64. For other Macs, choose amd64. For Linux, choose amd64. For Windows, choose amd64.

Install the `colonies` and `pollinator` binaries by moving them to a directory in your PATH. On Linux and Mac, you can move the binaries to `/usr/local/bin`. On Windows, you can move the binaries to `C:\Windows\System32`.

```
tar -xvf colonies_v1.7.12_linux_amd64.tar.gz
tar -xvf pollinator_v1.0.4_linux_amd64.tar.gz
sudo cp colonies_v1.7.12_linux_amd64/colonies /usr/local/bin
sudo cp pollinator_v1.0.4_linux_amd64/pollinator /usr/local/bin
```

IMPORTANT: On Mac, the `colonies` and `pollinator` binaries will be blocked the first time they are started since they are not downloaded from AppStore nor signed by an registered developer. They can be unblocked by clicking the `Open Anyway` button in `Privacy & Security` settings. The button is available for about an hour after you try to start the colonies or pollinator commands.


"colonies" can't be opened because Apple cannot check it for malicious software.
This software needs to be updated. Contact the developer for more information.
Chrome downloaded this file today at 10:28.
Show in Finder
OK


Privacy & Security
Search
Johan Kristiansson
Apple ID
Family
Wi-Fi
Bluetooth
Network
Notifications
Sound
Focus
Screen Time
General
Appearance
Accessibility
Control Center
Siri & Spotlight
Privacy & Security
Desktop & Dock
Displays
Wallpaper
Developer Tools
Sensitive Content Warning
Off
Analytics & Improvements
Apple Advertising
Security
Allow applications downloaded from
App Store
App Store and identified developers
"colonies" was blocked from use because it is not from an identified developer.
Allow Anyway
Allow accessories to connect
Ask for New Accessories
FileVault
On
Lockdown Mode
Off
Others

Verify installation

You also need valid creadentials to access the Colonies server (an env files). Or, you can also set up your own Colonies server by following the instructions here.

Verify that the installation was successful by running the following command:

```
source my-env-file
colonies executor ls
```

You should see a list of executors in the colony.

```
┌──────────────────┬──────────────────────┬──────────────────┬──────────────────────┐
│ NAME             │ TYPE                 │ LOCATION         │ LAST HEARD FROM      │
├──────────────────┼──────────────────────┼──────────────────┼──────────────────────┤
│ icekube          │ container-executor   │ RISE, Sweden     │ 2024-02-28 21:11:34  │
│ dev              │ container-executor   │ Rutvik, Sweden   │ 2024-02-28 21:11:35  │
│ leonardo-booster │ container-executor   │ Cineca, Italy    │ 2024-02-27 18:50:11  │
│ lumi-std         │ container-executor   │ CSC, Finland     │ 2024-02-28 21:11:43  │
└──────────────────┴──────────────────────┴──────────────────┴──────────────────────┘
```

To run an exector (optional)

To run your own executor, you also need to install Docker. On Linux (e.g Ubuntu), Docker can be installed by running the following commands:

```
sudo apt-get update
sudo apt-get install docker.io
```

You may also want to set permissions to run Docker without sudo.

```
sudo usermod -aG docker $USER
```

On Mac and Windows, Docker can be installed by downloading the Docker website website

Getting started

The Colonies CLI is a command line tool that allows you to interact with the Colonies API. It is a powerful tool that can be used to execute functions, manage processes, or deploy executors in a colony. To use the Colonies CLI, you first need to export several environmental variables.

```
export COLONIES_SERVER_TLS="true"
export COLONIES_SERVER_HOST="server.colonyos.io"
export COLONIES_SERVER_PORT="443"
export COLONIES_COLONY_NAME="hpc"
export
COLONIES_PRVKEY="e7957ca33481ce5cebc2571dea98da32d24fbe3db2d6d0916ec0165a26292299"
export COLONIES_EXECUTOR_NAME="johan-laptop"
export EXECUTOR_FS_DIR="$HOME/.colonies/cfs"
export EXECUTOR_PARALLEL_CONTAINERS="true"
export EXECUTOR_GPU="true"
export AWS_S3_ENDPOINT="s3.colonyos.io:443"
export AWS_S3_ACCESSKEY="accesskey"
export AWS_S3_SECRETKEY="secretkey"
export AWS_S3_REGION_KEY=""
export AWS_S3_BUCKET="hpc"
export AWS_S3_TLS="true"
export AWS_S3_SKIPVERIFY="false"
```

```
source env_file
```

The Colonies CLI has several subcommands. It always possible to get more help by adding the *–help* flag to the command, for example:

```
colonies --help
```

```
Colonies CLI tool

Usage:
  colonies [command]

Available Commands:
  attribute   Manage process attributes
  cluster     Manage clusters
  colony      Manage colonies
  completion  Generate the autocompletion script for the specified shell
  config      Show currently used configuration
  cron        Manage cron
  database    Manage internal database
  dev         Start a development server
  executor    Manage executors
  fs          Manage file storage
  function    Manage functions
  generator   Manage generators
  help        Help about any command
  key         Manage private keys
  log         Manage logging
  monitor     Manage Prometheus monitoring
  process     Manage processes
  server      Manage production server
  user        Manage users
  workflow    Manage workflows

Flags:
  -h, --help              help for colonies
      --insecure          Disable TLS and use HTTP
      --skip-tls-verify   Skip TLS certificate verification
  -v, --verbose           Verbose (debugging)

Use "colonies [command] --help" for more information about a command.
```

Or, to get help about the function subcommand.

```
colonies function --help
```

```
Manage functions

Usage:
  colonies function [command]

Available Commands:
  exec        Execute a Function
  ls          List all Functions
  register    Register a Function to an Executor
  remove      Remove a Function from an Executor  Hint: use 'colonies executor ls --
full' to get the functionid
  submit      Submit a Function specification

Flags:
  -h, --help   help for function

Global Flags:
      --insecure          Disable TLS and use HTTP
      --skip-tls-verify   Skip TLS certificate verification
  -v, --verbose           Verbose (debugging)

Use "colonies function [command] --help" for more information about a command.
```

Executing functions

Let's list all executors in the available in the colony. The colony is distributed network of executors running somehwere on the Internet. An executor is responsible for executing functions.

```
colonies executor ls
```

```
┌──────────────────┬──────────────────┬──────────────┬──────────────────────┐
│ NAME             │ TYPE             │ LOCATION     │ LAST HEARD FROM      │
├──────────────────┼──────────────────┼──────────────┼──────────────────────┤
│ leonardo-booster │ container-executor │ Cineca, Italy  │ 2024-02-28 11:28:11 │
│ icekube          │ container-executor │ RISE, Sweden   │ 2024-02-28 11:27:06 │
│ dev              │ container-executor │ Rutvik, Sweden │ 2024-02-28 11:27:19 │
│ lumi-std         │ container-executor │ CSC, Finland   │ 2024-02-28 11:28:00 │
└──────────────────┴──────────────────┴──────────────┴──────────────────────┘
```

One way of executing a function is to submit a function specification. The example below runs the command *echo Hello, World* in a container based on *ubuntu:20.04* on the LUMI supercomputer. The function is allowed to use 10GiB of memory and 1 CPU core.

```
{
    "conditions": {
        "executortype": "container-executor",
        "executornames": [
            "lumi-std"
        ],
        "nodes": 1,
        "processes-per-node": 1,
        "mem": "10Gi",
        "cpu": "1000m",
        "gpu": {
            "count": 0
        },
        "walltime": 60
    },
    "funcname": "execute",
    "kwargs": {
        "cmd": "echo Hello, World",
        "docker-image": "ubuntu:20.04"
    },
    "maxexectime": 55,
    "maxretries": 3
}
```

```
colonies function submit --spec hello.json  --follow
```

Depending on the load on the LUMI supercomputer, the process may take a few minutes to start. The *–follow* flag will print the logs from the process as soon as they are available.

```
INFO[0000] Process submitted
ProcessId=ad733c56110d444f9f98bfbfa9d96576039c4829a652c2307b86311650075fc3
INFO[0000] Printing logs from process
ProcessId=ad733c56110d444f9f98bfbfa9d96576039c4829a652c2307b86311650075fc3
Hello, World
INFO[0165] Process finished successfull
ProcessId=ad733c56110d444f9f98bfbfa9d96576039c4829a652c2307b86311650075fc3
```

Running a local executor

Docker compose can be used to run a local executor.

```
source env
mkdir -p ~/colonies/cfs
git clone https://github.com/colonyos/executors
cd executors/docker
docker-compose up
```

```
Creating docker_executor ... done
Attaching to docker_executor
docker_executor    | time="2024-02-28T14:27:48Z" level=error msg="Failed to set
location long"
docker_executor    | time="2024-02-28T14:27:48Z" level=error msg="Failed to set
location long"
docker_executor    | time="2024-02-28T14:27:49Z" level=info msg=Self-registered
ColonyName=hpc ExecutorName=johan-laptop
docker_executor    | time="2024-02-28T14:27:49Z" level=info msg="Docker Executor
started" ColoniesInsecure=false ColoniesServerHost=server.colonyos.io
ColoniesServerPort=443 ColonyName=hpc ColonyPrvKey="**********************"
ExecutorId=c6ffb4074f7618659eb5fa00040059a4aed5f16277b0520885809d2f793af532
ExecutorName=johan-laptop ExecutorPrvKey="**********************"
ExecutorType=container-executor FsDir=/home/johan/.colonies/cfs GPU=false HardwareCPU=
HardwareGPUCount=0 HardwareGPUMemory= HardwareGPUName= HardwareGPUNodesCount=0
HardwareMemory= HardwareModel=n/a HardwareNodes=1 HardwareStorage= K8sNamespace=
K8sPVC= Latitude=0 LocationDesc=n/a Longitude=0 ParallelContainers=false
SoftwareName="colonyos/dockerexecutor:v1.0.1" SoftwareType=docker
SoftwareVersion="colonyos/dockerexecutor:v1.0.1" Verbose=true
```

```
colonies executor ls
```

```
┌─────────────────┬──────────────────┬──────────────┬─────────────────────┐
│ NAME            │ TYPE             │ LOCATION     │ LAST HEARD FROM     │
├─────────────────┼──────────────────┼──────────────┼─────────────────────┤
│ leonardo-booster │ container-executor │ Cineca, Italy │ 2024-02-27 18:50:11 │
│ lumi-std        │ container-executor │ CSC, Finland  │ 2024-02-28 15:27:46 │
│ johan-laptop    │ container-executor │ n/a           │ 2024-02-28 15:27:49 │
│ icekube         │ container-executor │ RISE, Sweden  │ 2024-02-28 15:28:07 │
│ dev             │ container-executor │ Rutvik, Sweden │ 2024-02-28 15:28:09 │
└─────────────────┴──────────────────┴──────────────┴─────────────────────┘
```

Handling data

Execution of functions often involves handling data. The Colonies CLI has a subcommand for managing file storage. The file storage is a distributed file system called Colony FS (CFS), and can be used to store input data, output data, and intermediate data. Data stored in CFS is access from all executors in the colony.

The command below list all labels.

```
colonies fs label ls
```

```
┌──────────────────────────────┬───────┐
│ LABEL                        │ FILES │
├──────────────────────────────┼───────┤
│ /water/Masks                 │ 2841  │
│ /water/Images                │ 2841  │
│ /water                       │ 1     │
└──────────────────────────────┴───────┘
```

Let’s create a new label and store a file in it.

```
mkdir myfiles
echo "hi!" > myfiles/hello.txt
colonies fs sync -l /myfiles -d myfiles
```

```
INFO[0000] Calculating sync plans
Analyzing /home/johan/dev/github/enccs/~ ... done!
INFO[0000] Sync plans completed                          Conflict resolution=replace-
remote Conflicts=0 Download=0 Upload=1
INFO[0000] Add --syncplan flag to view the sync plan in more detail

Are you sure you want to continue? (yes,no): yes
Uploading /myfiles                       ... done! [4B]
```

```
┌──────────────────────────────┬───────┐
│ LABEL                        │ FILES │
├──────────────────────────────┼───────┤
│ /water/Masks                 │ 2841  │
│ /water/Images                │ 2841  │
│ /water                       │ 1     │
│ /myfiles                     │ 1     │
└──────────────────────────────┴───────┘
```

```
Try to sync to another computer or another directory.
```

```
colonies fs sync -l /myfiles -d myfiles2
```

That’s great, but how do I use the data in a function? It possible to reference the data in the function specification. The remote executor will then automatically sync the data to the container before the function is executed. Let’s try that.

```
{
    "conditions": {
        "executortype": "container-executor",
        "executornames": [
            "icekube"
        ],
        "nodes": 1,
        "processes-per-node": 1,
        "mem": "10Gi",
        "cpu": "1000m",
        "gpu": {
            "count": 0
        },
        "walltime": 60
    },
    "funcname": "execute",
    "kwargs": {
        "cmd": "cat /cfs/myfiles/hello.txt",
        "docker-image": "ubuntu:20.04"
    },
    "fs": {
        "mount": "/cfs",
        "dirs": [
            {
                "label": "/myfiles",
                "dir": "/myfiles",
                "keepfiles": false,
                "onconflicts": {
                    "onstart": {
                        "keeplocal": false
                    },
                    "onclose": {
                        "keeplocal": true
                    }
                }
            }
        ]
    },
    "maxexectime": 55,
    "maxretries": 3
}
```

```
colonies function submit --spec cat.json  --follow
```

```
INFO[0000] Process submitted
ProcessId=d81e3ea76afd5d45902c494a77cf72ab6046e1cf8700e8ac36b6f5a7168a4bc4
INFO[0000] Printing logs from process
ProcessId=d81e3ea76afd5d45902c494a77cf72ab6046e1cf8700e8ac36b6f5a7168a4bc4
hi!
INFO[0013] Process finished successfully
ProcessId=d81e3ea76afd5d45902c494a77cf72ab6046e1cf8700e8ac36b6f5a7168a4bc4
```

Nice, the function executed the command `cat /cfs/myfiles/hello.txt` and printed the content of the file `hello.txt` to the console.

Let's explore a tool called Pollinator to avoid spending time on creating complex JSON files.

Pollinator

Pollinator is a tool that automatically sync a local file to CFS and create a function specification. It abscracts away the complexity of creating function specifications, making it possible to develop on a local computer while executing on a powerful supercomputer.

Let's create a new Pollinator project and use the ICE Kubernetes cluster for function execution.

```
mkdir myproject
cd myproject
pollinator new -n icekube
```

As you can see, a file called `project.yml` is created. Pollinator uses the `project.yml` file to generate function specifications. The `project.yml` file contains some generic configuration, e.g. how resources should be allocated. It also contains a reference to a file called `main.py`, which contains some Python code we would like to execute.

```
projectname: a79b82a96a5c132374b26beb78953112f084055e29b73d63fe95fcdce5c4981b
conditions:
  executorNames:
  - icekube
  nodes: 1
  processesPerNode: 1
  cpu: 1000m
  mem: 1000Mi
  walltime: 600
  gpu:
    count: 0
    name: ""
environment:
  docker: python:3.12-rc-bookworm
  rebuildImage: false
  init-cmd: pip3 install numpy
  cmd: python3
  source: main.py
```

Also, notice that a directory called `cfs` is created. The `cfs` directory contains three subdirectories:

- src
- result
- data

The `src` directory is synchronized before the container starts. The `data` directory is also synchronized before the container starts, but not deleted after the container has run to completion. The `result` directory is synchronized after the container has finished. This is a useful place to store generated data, .e.g model data after training a neural network.

Let's run a simple hello world Python program on Kubernetes.

```
print("Hello, World")
```

```
echo 'print("Hello, World")' > cfs/src/main.py
```

```
pollinator run --follow
```

```
INFO[0000] Process submitted,
ProcessID=24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298
INFO[0000] Follow process at https://dashboard.colonyos.io/process?
processid=24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298
Collecting numpy
  Obtaining dependency information for numpy from
https://files.pythonhosted.org/packages/0f/50/de23fde84e45f5c4fda2488c759b69990fd4512387a
1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Downloading numpy-1.26.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 1.2 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(18.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 50.8 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.4
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager. It is recommended to use a
virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
Hello, World
INFO[0017] Process finished successfully
```

To run it on the LUMI supercomputer, just change the executor name in the project.yml file to `lumi-std` and run `pollinator run --follow` again.

We can also check the status of the process by typing:

```
colonies process get -p
24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298
```

Process

Id	24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298
IsAssigned	True
InitiatorID	bcaeac1a507036f7fed0be9d38c43ba973be7c0064d1b0b010ede2f088093b3f
Initiator	johan
AssignedExecutorID	ef9943aa7a7e9aec2e00bac8a739fa5886d9df8fe648349596b44054e18d9d7c
AssignedExecutorID	Successful
PriorityTime	1708712143825558275
SubmissionTime	2024-02-28 19:15:43
StartTime	2024-02-28 19:15:43
EndTime	2024-02-28 19:15:43
WaitDeadline	0001-01-01 00:53:28
ExecDeadline	2024-02-28 19:25:42
WaitingTime	35.886ms
ProcessingTime	16.542659s
Retries	0
Input	
Output	
Errors	

Function Specification

Func	execute
Args	None
KwArgs	init-cmd:pip3 install numpy rebuild-image:false ar...
MaxWaitTime	-1
MaxExecTime	599
MaxRetries	3
Label	test_label

Conditions

Colony	hpc
ExecutorNames	icekube
ExecutorType	container-executor
Dependencies	
Nodes	1
CPU	1000m
Memory	1000Mi
Processes	0
ProcessesPerNode	1

```
| Storage          | 0Mi                |
| Walltime         | 600                |
| GPUName          |                    |
| GPUs             | 0                  |
| GPUPerNode       | 0                  |
| GPUMemory        | 0Mi                |
└──────────────────┴────────────────────┘

┌──────────────────────────────────────────────────────────────────────────────────────────────────────────┐
| Attributes
|
├──────────────────────────────────────────────────────────────────┬──────────────┬──────────────────────┐
| ATTRIBUTEID                                                      | KEY          | TYPE
|     |
├──────────────────────────────────────────────────────────────────┼──────────────┼──────────────────────┤
| 652d5fbe8028b99c9e9bccce9ed9e6bd7846a6a569277b0ca3dc4edf05383e16 | PROJECT_DIR |
/cfs/pollinator/a79b82a96a5c13... | Env |
└──────────────────────────────────────────────────────────────────┴──────────────┴──────────────────────┘
```

```
colonies log get -p 24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298
```

If we want to see the logs from the process, we can use the *colonies log get* command.

```
Collecting numpy
  Obtaining dependency information for numpy from
https://files.pythonhosted.org/packages/0f/50/de23fde84e45f5c4fda2488c759b69990fd4512387a
1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Downloading numpy-1.26.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 1.2 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(18.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 50.8 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.4
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager. It is recommended to use a
virtual environment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
Hello, World
```

If we don't know the process ID, we can use the `colonies log search` command to search for logs.

```
colonies log search --text "Hello, World"
```

```
┌──────────────┬──────────────────────────────────────────────────────────────────┐
│ Timestamp    │ 2024-02-28 11:37:13                                              │
│ ExecutorName │ lumi-std                                                         │
│ ProcessID    │ ad733c56110d444f9f98bfbfa9d96576039c4829a652c2307b86311650075fc3 │
│ Text         │ Hello, World                                                     │
└──────────────┴──────────────────────────────────────────────────────────────────┘

┌──────────────┬──────────────────────────────────────────────────────────────────┐
│ Timestamp    │ 2024-02-28 19:15:58                                              │
│ ExecutorName │ icekube                                                          │
│ ProcessID    │ 24519ebe1d97c0627c971623e33e4a4963f1d8d55920c1a0437b4ad12f3be298 │
│ Text         │ Hello, World                                                     │
└──────────────┴──────────────────────────────────────────────────────────────────┘
```

Introduction

In this tutorial, we will train a machine learning model to identify water in Sentinel-2 satellite images. We will be using code from this GitHub repo using this dataset.

First, find a target executor.

```
┌──────────────────┬────────────────────┬──────────────┬─────────────────────┐
│ NAME             │ TYPE               │ LOCATION     │ LAST HEARD FROM     │
├──────────────────┼────────────────────┼──────────────┼─────────────────────┤
│ icekube          │ container-executor │ RISE, Sweden │ 2024-02-28 20:05:45 │
│ dev              │ container-executor │ Rutvik, Sweden │ 2024-02-28 20:05:45 │
│ leonardo-booster │ container-executor │ Cineca, Italy │ 2024-02-27 18:50:11 │
│ lumi-std         │ container-executor │ CSC, Finland │ 2024-02-28 20:06:01 │
└──────────────────┴────────────────────┴──────────────┴─────────────────────┘
```

Generate an empty working, targeting the ICEKube K8s cluster. Note that the target executor can be changed later.

```
mkdir waterml
cd waterml
pollinator new -n icekube
```

```
INFO[0000] Creating directory                            Dir=./cfs/src
INFO[0000] Creating directory                            Dir=./cfs/data
INFO[0000] Creating directory                            Dir=./cfs/result
INFO[0000] Generating                                    Filename=./project.yaml
INFO[0000] Generating                                    Filename=./cfs/data/hello.txt
INFO[0000] Generating                                    Filename=./cfs/src/main.py
```

Dataset

Copy the `water_body_dataset` to the `./cfs/data` directory.

```
cp ~/water_body_dataset ./cfs/data/water
```

If the dataset is already stored in Colonies CFS, you can copy the dataset directly from CFS to the project directory.

```
colonies fs sync -l /water -d ./cfs/data/water
```

The dataset will upload next time the project run and will be available in the container at these directories:

```
projdir = os.environ.get("PROJECT_DIR")
image_path = projdir + '/data/water/Images/'
mask_path = projdir + '/data/water/Masks/'
```

Build a Docker container (optional)

We are going the Container Executor, which comes in three variants.

1. **Kube Executor** runs containers as Kubernetes batch jobs.
2. **Docker Executor** runs containers as Docker containers on a baremetal servers or VMs.
3. **HPC Executor** runs containers as Singularity containers on HPC systems, managing them as Slurm jobs.

As the *function specification* is identical, meaning that we can easily switch between these 3 types of executors. To run containers, we first need to create a Dockerfile with the following content:

```
FROM docker.io/tensorflow/tensorflow:2.13.0-gpu

RUN apt-get update && apt-get install -y python3 python3-pip wget vim git fish libgl1-
mesa-glx libglib2.0-0
RUN python3 -m pip install --upgrade pip
RUN pip3 install pycolonies opencv-python tqdm Pillow scikit-learn keras matplotlib
numpy
```

Build and publish the Dockerfile and publish the Docker image at public Docker registry.

```
docker build -t johan/hackaton .
docker push johan/hackaton
```

The `johan/hackaton` Docker image has already been published at DockerHub.

Training the model

Now that we have prepared the dataset and created a Docker container, it's time to proceed with training the model.

Configure the Pollinator project

```
projectname: johantest
conditions:
  executorNames:
  - icekube
  nodes: 1
  processesPerNode: 1
  cpu: 10000m
  mem: 15000Mi
  walltime: 600
  gpu:
    count: 1
    name: "nvidia-gtx-2080ti"
environment:
  docker: johan/hackaton
  rebuildImage: false
  cmd: python3
  source: main.py
```

Replace main.py

Download source code from this GitHub repo.

```
wget -O cfs/src/main.py
https://raw.githubusercontent.com/johankristianss/colonyoshackaton/main/src/main.py
```

Note that the Python code saves the training result and a random prediction example in the result directory, which is automatically synchronized back to the client after process completion.

```
plt.savefig(projdir + '/result/res_' + processid + '.png')
plt.savefig(projdir + '/result/samples_' + processid + '.png')
```

```
ls cfs/result
```

```
.rw-r--r--  55k johan 12 Dec 21:40
res_076e273a1d082dd2886892dfd7d1723e12c747cf2899f2c2ede27ceb55e06ae2.png
.rw-r--r-- 266k johan 12 Dec 21:40
samples_076e273a1d082dd2886892dfd7d1723e12c747cf2899f2c2ede27ceb55e06ae2.png
```

Train the model

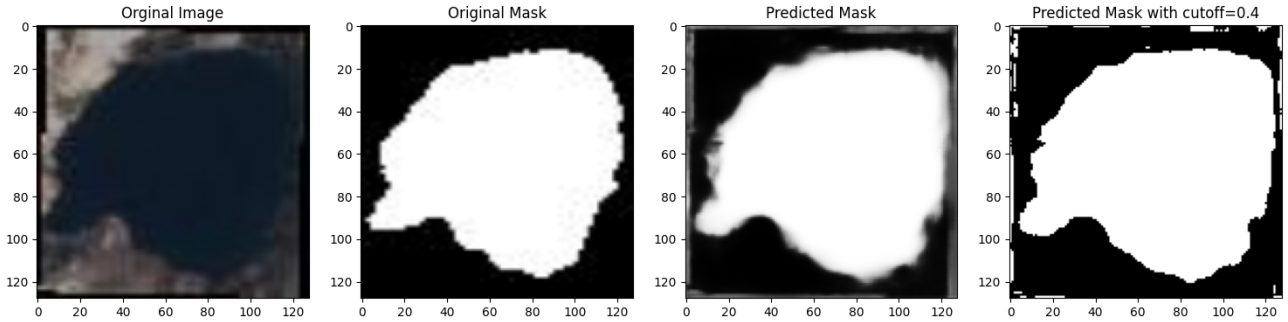
Pollinator will automatically synchronize the `cfs/src`, `cfs/data`, and `cfs/result` directories to Colonies CFS, generate a *function specification* and then submit the *function specification*, follow the process execution, and upon completion, synchronize the project files back to your local computer.

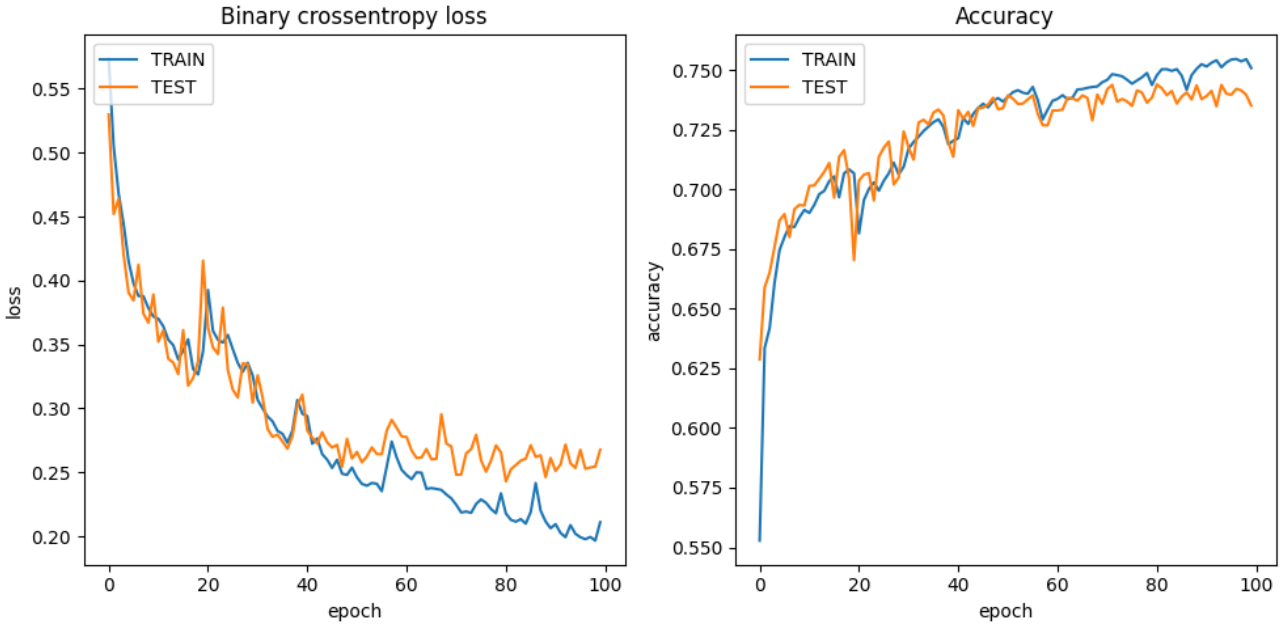
```
pollinator run --follow
```

```
67/67 [==============================] - 1s 18ms/step - loss: 0.3434 - accuracy: 0.7024
- val_loss: 0.3263 - val_accuracy: 0.7038
Epoch 25/30
67/67 [==============================] - 1s 17ms/step - loss: 0.3307 - accuracy: 0.7092
- val_loss: 0.3146 - val_accuracy: 0.7121
Epoch 26/30
67/67 [==============================] - 1s 18ms/step - loss: 0.3139 - accuracy: 0.7140
- val_loss: 0.2947 - val_accuracy: 0.7249
Epoch 27/30
67/67 [==============================] - 1s 17ms/step - loss: 0.3226 - accuracy: 0.7110
- val_loss: 0.3027 - val_accuracy: 0.7244
Epoch 28/30
67/67 [==============================] - 1s 17ms/step - loss: 0.2994 - accuracy: 0.7208
- val_loss: 0.2910 - val_accuracy: 0.7259
Epoch 29/30
67/67 [==============================] - 1s 17ms/step - loss: 0.2910 - accuracy: 0.7239
- val_loss: 0.2781 - val_accuracy: 0.7261
Epoch 30/30
67/67 [==============================] - 1s 17ms/step - loss: 0.2856 - accuracy: 0.7258
- val_loss: 0.2733 - val_accuracy: 0.7313
23/23 [==============================] - 0s 4ms/step

INFO[0141] Process finished successfully
ProcessID=61e597845ed3df4456c5be7d358e35141b8dc4c1f76a89d7caad0f31f792106c
Downloading
samples_076e273a1d082dd2886892dfd7d1723e12c747cf2899f2c2ede27ceb55e06ae2.png 100%
[===============] (5.0 MB/s)
Downloading res_076e273a1d082dd2886892dfd7d1723e12c747cf2899f2c2ede27ceb55e06ae2.png
100% [===============] (1.7 MB/s)
```

We can now open the sample and training plot pictures.





Quick Reference

Instructor's guide

Why we teach this lesson

Intended learning outcomes

Timing

Preparing exercises

e.g. what to do the day before to set up common repositories.

Other practical aspects

Interesting questions you might get

Typical pitfalls

Who is the course for?

This course is for people who want to learn about more about ColonyOS and how to use it.

About the course

This course is an introduction to ColonyOS. It is designed to be a hands-on course, where you will learn by doing. The course is divided into three parts:

- Introduction
- Tutorial - Image Processing
- Tutorial - How implement a custom Executor

See also

Credits

The lesson file structure and browsing layout is inspired by and derived from work by CodeRefinery licensed under the MIT license. We have copied and adapted most of their license text.

Instructional Material

- **No additional restrictions** - You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

With the understanding that:

- You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.
- No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Software

Except where otherwise noted, the example programs and other software provided with this repository are made available under the OSI-approved MIT license.