Actions

Welcome to our step-by-step guide on creating an Action! This tutorial is designed to walk you through the process of developing and implementing an action, a fundamental building block in the Giza workflow system.

By the end of this tutorial, you'll have a solid understanding of how to leverage Giza's powerful features to streamline your workflows and create powerful Verifiable ML solutions. Let's get started!

Writing an Action

The@action decorator is used to define an Action:

```

Copy fromgiza_actions.actionimportaction

@action defmy_action(): return

```

Within an action definition, you have the flexibility to include any code, as long as it's valid Python - there are no strict guidelines to follow.

Each action is distinguished by its name. You have the option to assign a specific name to the action using the name parameter. If you choose not to provide a name, the Actions SDK automatically adopts the name of the action function.

```

Copy @action defmy_action(): return

```

Now, you need to create all thetasks that are going to be inside your Action. Tasks are defined as Python functions and use the@task decorator. Tasks can have any size and be contained in external modules. We recommend to make tasks as atomic as possible.

Actions can call tasks to allow Giza to orchestrate and track more granular units of work:

```

Copy fromgiza_actions.actionimportaction

@task defprint_hello(name): print(f"Hello{name}!")

@action defhello_world(name="world"): print_hello(name)

```

Congratulations! You created your first Action. Now go to thedeployments how-to guide to learn how to deploy your Action

Writing an Action with Parameters

Actions can be called with both positional and keyword arguments. These arguments are resolved at runtime into a dictionary of parameters mapping name to value:

```

Copy fromgiza_actions.actionimportAction,action fromgiza_actions.taskimporttask

@task defpreprocess(example_parameter:bool=False): print(f"Preprocessing with example={example_parameter}") print("Preprocessing...")

@task deftransform(example_parameter:bool=False): print(f"Transforming with example={example_parameter}") print(f"Transforming...")

@action(log_prints=True) definference(example_parameter:bool=False): print(f"Running inference with example={example_parameter}") preprocess(example_parameter=example_parameter) transform(example_parameter=example_parameter)

```

This action includes a parameterexample request , which allows for further customization and parameterized runs, which greatly improves the development opportunities around the actions. This parameter then can be used by tasks created for the action. This is easily achievable by just adding the parameter to the decorated function as a normal function parameter.

Writing Scheduled Actions

Schedules show how to create new actions runs for you automatically on a specified cadence (interval) or cron expression. This is accomplished when serving the action.

Using Cron

Add a valid cron expression using thecron parameter. Multiple online tools can be used to generate such expressions, likehttps://crontab.cronhub.io/

```
```

Copy action_deploy=Action(entrypoint=inference, name="inference-local-action")

This will execute every minute

action_deploy.serve(name="inference-local-action", cron=" * * *")

```

Using an interval

Specify an interval, defaults to seconds

```

Copy fromdatetimeimporttimedelta

Define tasks and action

...

Using a number

action_deploy=Action(entrypoint=inference, name="inference-local-action")

Executes each 10 seconds

action_deploy.serve(name="inference-local-action", interval=10)

```

Previous Workspaces Next Deployments

Last updated1 day ago
```