Distributed Computation with Python and Celery

Hi

Lars.Butler@gmail.com

www.openquake.org

Overview

Part 1:

- What is Celery?
- How is it useful?
- How does it work?

Part 2:

- What is a 'task'?
- What are the supported message brokers?
- How are task results stored?
- Configuring Celery

Demo:

- Architecture examples
- Code examples

Questions

Part 1

Conceptual stuff

What is Celery?

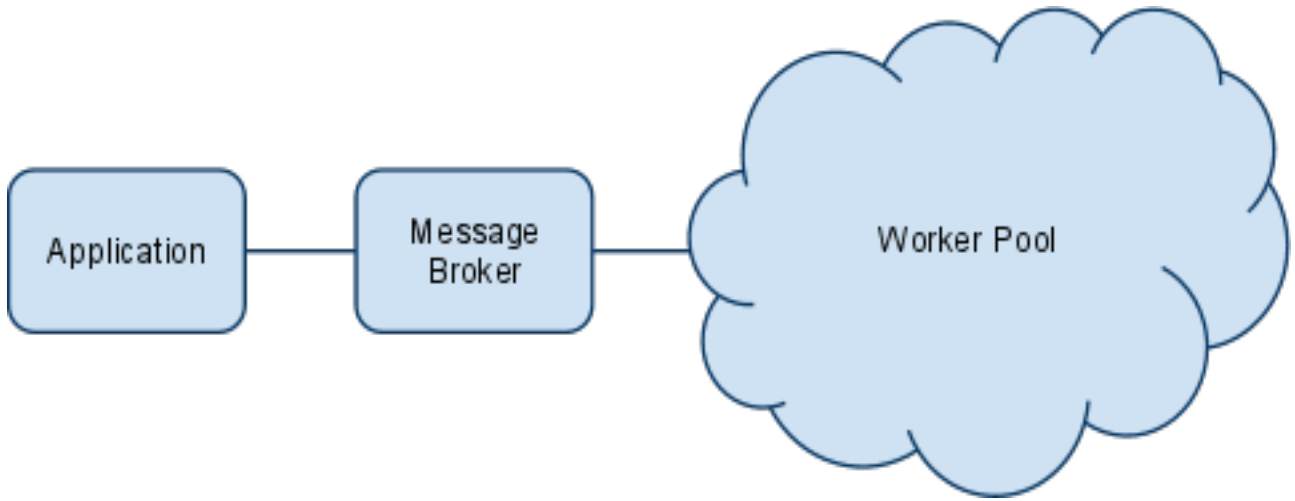
- A delicious vegetable
- An asynchronous task queue based on distributed messaging

Homepage: http://celeryproject.org
Source code: http://pypi.python.org/pypi/celery#downloads
Revision control: https://github.com/ask/celery
License: BSD-style (https://github.com/ask/celery/blob/master/LICENSE)
Documentation: http://docs.celeryproject.org/

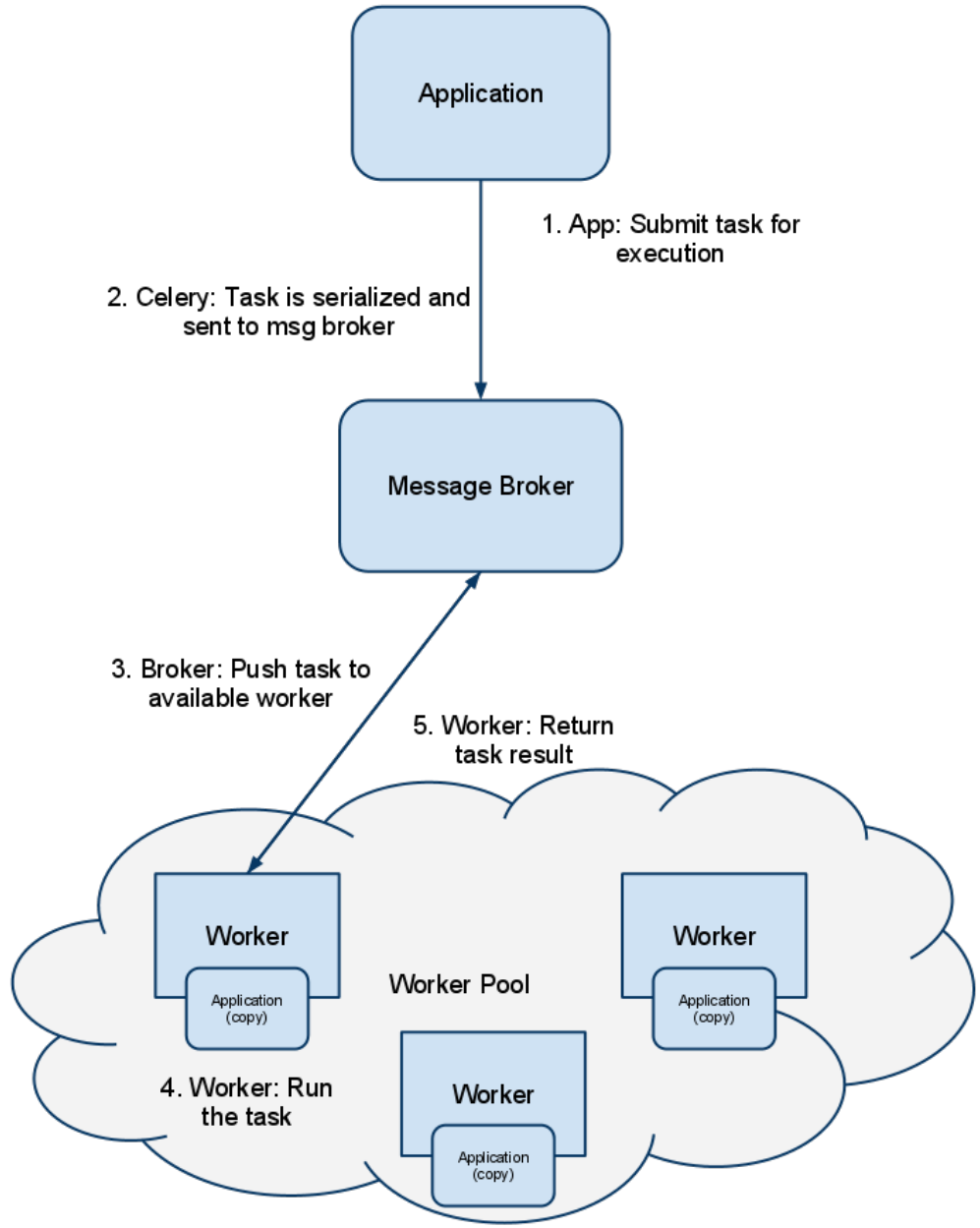
Version 1.0: February 10th 2010
Version 2.2: February 1st 2011 (latest version)

How is Celery useful?

- Break computations down into small chunks ('tasks')
- Process tasks using remote workers
- Encapsulates horizontal scaling concerns



How does it work?



Part 2

Details

What is a 'task'?

- A Celery task is a set of computations
  - Can be executed asynchronously
  - Can be purely procedural or return a result
- Tasks are functions or methods with the '@task' decorator from the celery.decorators package
- @task must be the 'outermost' decorator, like so:

```
@task
@foo
@bar
def some_task(x, y):
  # task code here
```

Example tasks

- Complex/time-consuming mathematical computations
- Image processing/rendering
- Searching/sorting large data sets
- Pathfinding algorithms on large data sets
- Servicing a client request in a high-availability system
- Virtually anything that requires a non-trivial amount of time or processing power

What are the supported message brokers?

Preferred:

- RabbitMQ

Limited support:

- Redis
- Beanstalk
- MongoDB
- CouchDB
- RDBMSs (using SQLAlchemy or Django ORM)
- Others

How are task results stored?

- Result backends supported:
  - database (default)
  - MongoDB
  - Redis
  - AMQP
  - Others
- If task result data is of a non-trivial size, it may be useful to store the data produced by the task in a database or key value store.

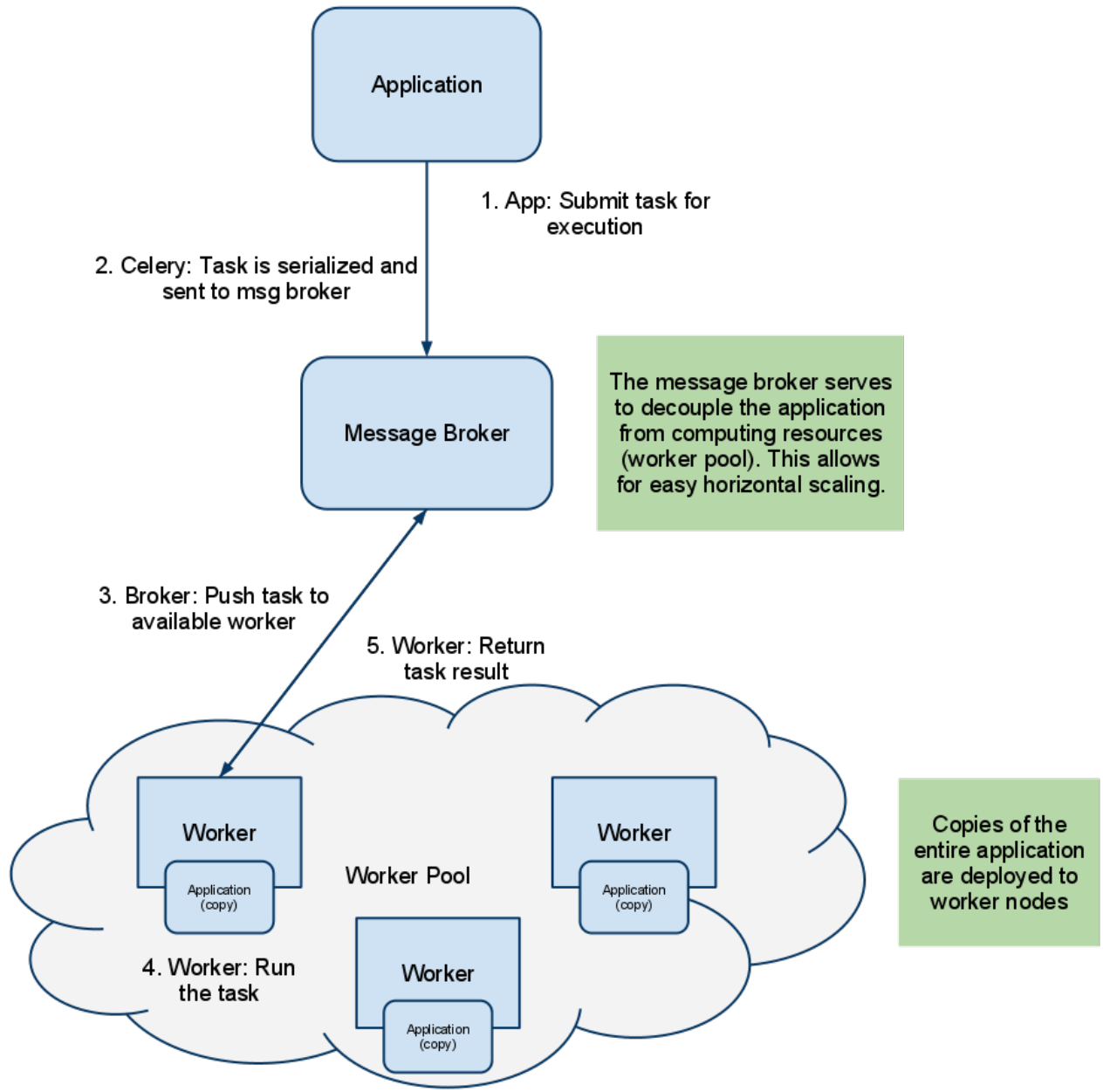
Configuring Celery: AMQP

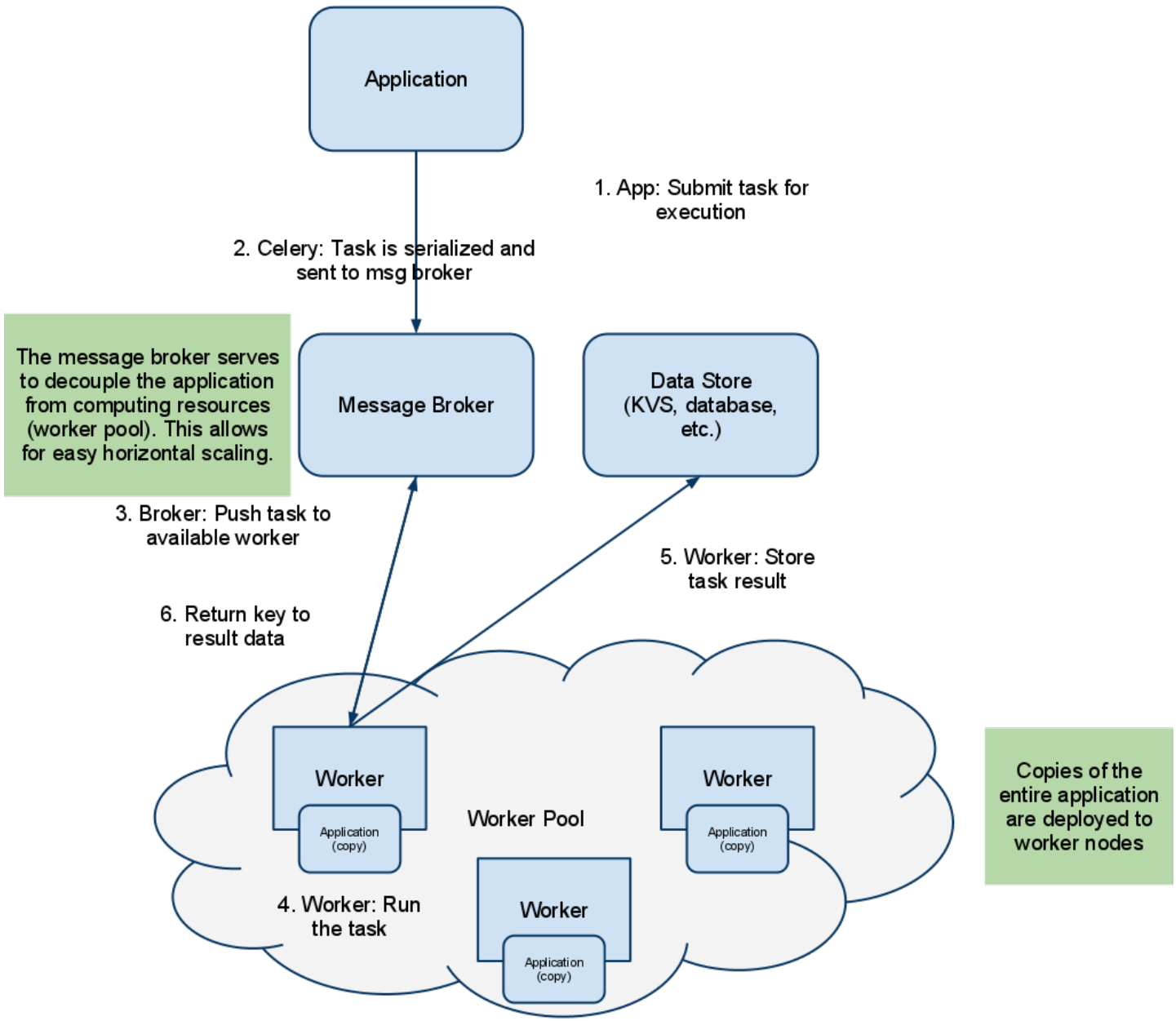
Example celeryconfig.py:

```
import sys

sys.path.append('.')

BROKER_HOST = "localhost"
BROKER_PORT = 5672
BROKER_USER = "celeryuser"
BROKER_PASSWORD = "celery"
BROKER_VHOST = "celeryvhost"

CELERY_RESULT_BACKEND = "amqp"

CELERY_IMPORTS = ("tasks",)
```

Configuring Celery: SQLite

```
import sys

sys.path.append('.')

BROKER_BACKEND = "sqlalchemy"
BROKER_HOST = "sqlite:///backend.db"

# optional: "database" is the default result backend
# CELERY_RESULT_BACKEND = "database"
CELERY_RESULT_DBURI = "sqlite:///result.db"

# optional: enable verbose logging from SQLAlchemy.
# CELERY_RESULT_ENGINE_OPTIONS = {"echo": True}

CELERY_IMPORTS = ("tasks",)
```


Application
1. App: Submit task for execution
2. Celery: Task is serialized and sent to msg broker
Message Broker
The message broker serves to decouple the application from computing resources (worker pool). This allows for easy horizontal scaling.
3. Broker: Push task to available worker
5. Worker: Return task result
Worker
Application (copy)
Worker Pool
Worker
Application (copy)
4. Worker: Run the task
Worker
Application (copy)
Copies of the entire application are deployed to worker nodes


Application
1. App: Submit task for execution
2. Celery: Task is serialized and sent to msg broker
The message broker serves to decouple the application from computing resources (worker pool). This allows for easy horizontal scaling.
Message Broker
Data Store (KVS, database, etc.)
3. Broker: Push task to available worker
5. Worker: Store task result
6. Return key to result data
Worker
Application (copy)
Worker Pool
Worker
Application (copy)
4. Worker: Run the task
Worker
Application (copy)
Copies of the entire application are deployed to worker nodes

Code Examples

https://github.com/larsbutler/celery-examples

(These slides are in the repo as well.)