PL/Container Introduction
Customize and Secure the Runtime and Dependencies of Procedural Languages

Hubert Zhang (hzhang@pivotal.io)
Jack WU (jwu@pivotal.io)
Agenda

- The Problem
- What is PL/Container
- How to use PL/Container
- PL/Container Internals
- Future Work
- Q+A
The Problem
We generate More and More Data
We generate More and More Data

1.2 ZB
2010

2.8 ZB
2012

8.5 ZB
2015

40 ZB
2020

1 ZB = 1,000,000,000,000,000,000,000,000,000 Byte
We want to analyze data for knowledge
We want to analyse data IN Database
But...
PL/Python and PL/R are UNTRUSTED Languages
Only Superuser can Create UDF in Untrusted Languages

System("rm -rf /data")
The Problem: Triangle Dependency

1. Greenplum
2. Operation System
3. Python / R
4. TensorFlow
Resolve The Problem: untrusted -> untrusted

Data Scientist

Run UDF

Create UDF

DBA

Review & Create

Pivotal
How to Make untrusted to untrusted?

PL/Container

Pivotal
What is PL/Container
What is PL/Container?

PL/Container is a customizable, secure runtime for Greenplum Database Procedural Languages.

- Greenplum Database Extension
- Stateless
- Based on Docker Container
- Customizable
- Secure
- Isolated
How UDF run in PL/Container?

PL/Container is a customizable, secure runtime for Greenplum Database Procedural Languages.

a. PL/Container Extension starts a docker container (only in 1st call)
b. Transfer UDF and data to docker container
c. Run the UDF in docker container
d. Contact the docker container to get the results
How to use PL/Container
Install PL/Container on Greenplum

Install from Source Code

- source $GPHOME/greenplum_path.sh
- make install

Install from GPPKG

- gppkg -i plcontianer-1.1.0-rhel7-x86_64.gppkg
- no additional dependencies

Prerequisites

Platform
Centos 6.6+ or 7.x

Database
Greenplum 5.2+

Docker
Docker 17.05+ on Centos7
Docker 1.7+ on Centos6
Build Custom Docker Image (optional)

Minimum Requirement:

- Python or R environment
- Add location of libpython.so and libR.so to LD_LIBRARY_PATH

```
FROM continuumio/anaconda3
ENV LD_LIBRARY_PATH "/opt/conda/lib:$LD_LIBRARY_PATH"
```

Customize Your image:

- Install specific packages

```
FROM continuumio/anaconda3
RUN conda install -c conda-forge -y tensorflow
ENV LD_LIBRARY_PATH "/opt/conda/lib:/usr/local/lib:$LD_LIBRARY_PATH"
```
Configure PL/Container

**XML**

```
runtime
  <id>
  <image>
  <command>
  <shared directory>
  <setting memory_mb>
  <setting cpu_share>

container cgroup node
memory.memsw.limit_in_bytes
cpu.shares
```
Run PL/Container

Running a simple plpython UDF to calculate $\log_{10}$

```
postgres=# CREATE LANGUAGE plpythonu; plcontainer;

postgres=# CREATE OR REPLACE FUNCTION pylog10(input double precision) RETURNS double precision AS $$
import math
return math.log10(input)
$$ LANGUAGE plpythonu;

postgres=# SELECT pylog10(100);
    pylog10
----------
        2
(1 row)
```
Run PL/Container

Running a simple PL/Container UDF to calculate $\log_{10}$

```sql
postgres=# CREATE EXTENSION plcontainer;

postgres=# CREATE OR REPLACE FUNCTION pylog10(Input double precision) RETURNS double precision AS $$
# container: plc_python_shared
import math
return math.log10(input)
$$ LANGUAGE plcontainer;

postgres=# SELECT pylog10(100);
pylog10
----------
   2
(1 row)
```
How to use PLcontainer

Demos
PGConf 2018: PL/Container Introduction

PL/Container Internal
PL/Container Internals

- Message Protocol
- SPI Support
- Pluggable Backend
- Resource Management
- Error Handling
- Performance
Message Protocol

PL/Container use messages to communicate between QEs and containers.

- plcMsgPing
- plcMsgCallreq
- plcMsgResult
- plcMsgError
- plcMsgLog
- plcMsgSQL
- plcMsgSubtransaction
- plcMsgRaw
SPI support

Server Programming Interface enable UDF to run SQL queries.

**Problem:** SPI is called inside container but executed at QE side.

- **Container**
  - `plpy.execute(query)`
  - `plan=plpy.prepare` (Where Query Generated)
  - `plpy.execute(plan)` (Where Query Executed)

- **Query Executor**
  - `SPI_execute()` (Where Query Executed)
  - `create&cache plan`
  - `SPI_execute_plan`
Pluggable Backend

Separate Process
- GPDB Query Executor
  - C CODE
- Python Executor
  - Python Code

Docker as a sandbox

Container as a service

GPDB Query Executor

Pivotal
Resource Management

Container level

- Memory: memory limit, minimum is 4M
- CPUShares: relative weight of CPU

Extension level

- Integrated with GPDB resource group extension framework.
Resource Management

**Container level**
- Memory: memory limit, minimum is 4M
- CpuShares: relative weight of CPU

**Extension level**
- Integrated with GPDB resource group extension framework.

```plaintext
create resource group plgroup
  (concurrency=0,
   cpu_rate_limit=10,
   memory_limit=30,
   memory_auditor='cgroup')
```
Resource Management

Container level
- Memory: memory limit, minimum is 4M
- CpuShares: relative weight of CPU

Extension level
- Integrated with GPDB resource group extension framework.

```sql
create resource group plgroup (concurrency=0, cpu_rate_limit=10, memory_limit=30, memory_auditor='cgroup')
```
Error Handling

Container failure should not affect GPDB core
- Containers fail to create
- Containers fail to start
- Containers crash when running
- Cached containers crash

Container cleanup
- Query Cancel
- QE error
- Cached QE quit when idle for a long time (By Cleanup process)
Performance

Optimization
- Cached container (lifecycle same as QE)
- Unix domain socket
- Type conversion
- Resource management (CPU share)

Best practices
- Array instead of multiple rows.
- Complex UDF instead of simple one
Performance

Test Environment

- Hardware: 6 virtual machines, each with 19G memory and 5 processors. (Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz)
- Software: Centos7, GPDB 5.2 with 30 segments.

Workloads

- Long-running function
- Large input array function
- Large output array function
Performance

Long-running function

CREATE OR REPLACE FUNCTION pysleep(i int) RETURNS void AS $$
# contaziner: plc_python_shared
import time
Time.sleep(i)
$$ LANGUAGE plcontainer;

SELECT count(pysleep(1)) FROM tbl;

no performance downgrade
Performance

Large input array function

CREATE OR REPLACE FUNCTION pylargeint8in(a int8[]) RETURNS float8 AS $$
#container : plc_python_shared
return sum(a)/float(len(a))
$$ LANGUAGE plcontainer;

SELECT count(pylargeint8in(ARRAY(SELECT column1 FROM tbl1))) FROM tbl2;

2 times performance downgrade for Python
30% performance improvement for R
Performance

Large output array function

CREATE OR REPLACE FUNCTION pylargeoutfloat8(num int) RETURNS float8[] AS $$
# container: plc_python_shared
return [x/3.0 for x in range(num)]
$$ LANGUAGE plcontainer;

SELECT count(pylargeoutfloat8(n)) FROM tbl;

4 times performance improvement
Future Work
PL/Container Future Work (subject to change)

- Container Orchestrator / Cloud
  - kubernetes
  - Pivotal Container Service™

- Support More Languages
  - C Programming
  - Python

- Support More Technology
  - TensorFlow
  - Anaconda
  - Keras