Elasticity Tool-Catalog Integration Design Document

Design for integration of the Elasticity module and Tool Catalog

1. Goal
To integrate the tool catalog and the elasticity module, so in the Tool Catalog, an admin user can deploy an approved tool or tools to a certain DTS instance. The current use case is to deploy Docker images. For VM images, they need to be manually created, tested, and uploaded to the managed cloud. This integration also supports it.

Abbreviations used in this document: the Elasticity Module (EM), Tool Catalog (TC).

2. Design Considerations
2.1 Docker image creation steps
Typical steps from a Docker file -> a working Docker image at the Docker Hub:

1. build a Docker image from the Docker file;
2. test the Docker image -- requires manual testing and judgment;
3. upload to the Docker Hub.

Step 2 requires human intervention, not fully automated.

2.2 VM image creation steps

Typical steps to create a working VM image:

1. start from a base VM, make modifications, then save a snapshot of the VM as an image;
2. test the VM image -- manually.
3. upload the image to the project if it's done at another cloud. If step 1 and 2 above are done in the same cloud and project, then this step can be skipped.

2.3 Information needed by EM from TC to deploy a tool

EM needs the following info to deploy tools. The Tool Catalog needs to provide them to EM:

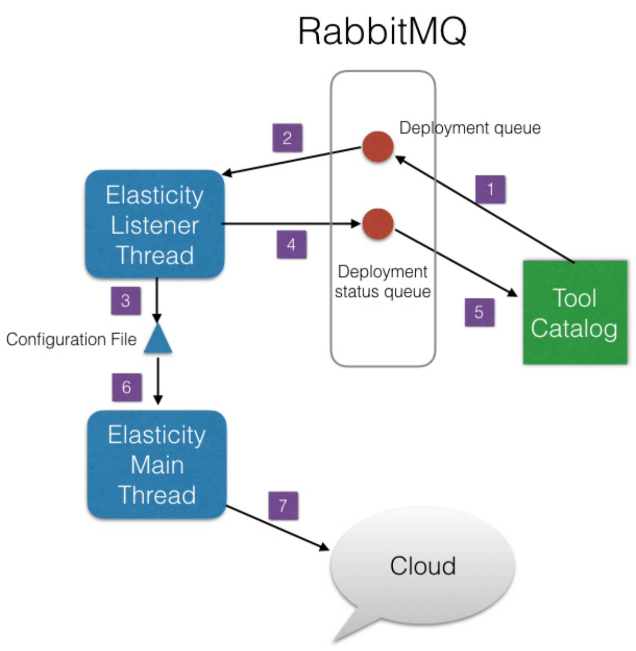
- RabbitMQ info: protocol, host name, port number, virtual host.
  (Example: http://rabbitmq.ncsa.illinois.edu:15672/prod)

  This is to uniquely identify the DTS/DAP instance to deploy to, not to access them. So it does not need the user name/password.

- For Docker, working Docker image names uploaded to the Docker Hub, min # of instances to deploy, the extractor name, and a container naming prefix.

- For VM images, image names uploaded into the cloud, min # of instances to deploy, upstart service name, flavor name (hardware configuration), SSH key pair name, SSH user name, extractors contained in the image, and an instance naming prefix.

2.4 Communication Design



1. TC: create a page to allow an admin user to enter the information (RabbitMQ access and Docker/VM image information).

2. Use RabbitMQ as a communication medium between TC and EM.

   The following are the communication work flow:

   1. EM spawns a thread to listen to a deployment queue in RabbitMQ (such as "elasticity-deploy");

   2: TC: an admin user uses a TC page to deploy a tool. After choosing one or multiple tools, the user clicks a button (such as "Deploy").

   3. TC sends a message to the deployment queue ("elasticity-deploy").

   4. The EM listening thread decides and updates the EM config file if needed; if the config file is updated, the main thread reloads the config and deploys the tools.

   Nice to have:
   5. The EM listening thread sends a message to a deployment status queue ("elasticity-deploy-status") in RabbitMQ.

   6. TC: create a listener to listen to the deployment status queue. The callback function saves the status in a collection in MongoDB.

   7. TC: create a page to display the deployment statuses.

3. Use JSON as the exchange data format.
   It is well supported in Python (EM) and Scala (TC) and we are familiar with handling the format.

This design has the following advantages:
- only on-demand traffic is generated;
- integration with TC is de-coupled from EM's main thread -- the EM
  main thread only checks whether the config file is updated;
- it does not need to create another web service, thus reduces the
  related complexity.

It seems to be the best choice among the alternatives we have considered.
Using it, we will need to write new code in TC to communicate with
RabbitMQ for publishing and consuming messages, but we can reuse some
code in Clowder's RabbitMQ plugin.