



Public Cloud - Azure workshop

Orchestrating and configuring workloads in Azure

By Marco Berube
February 2017

 @mberube9

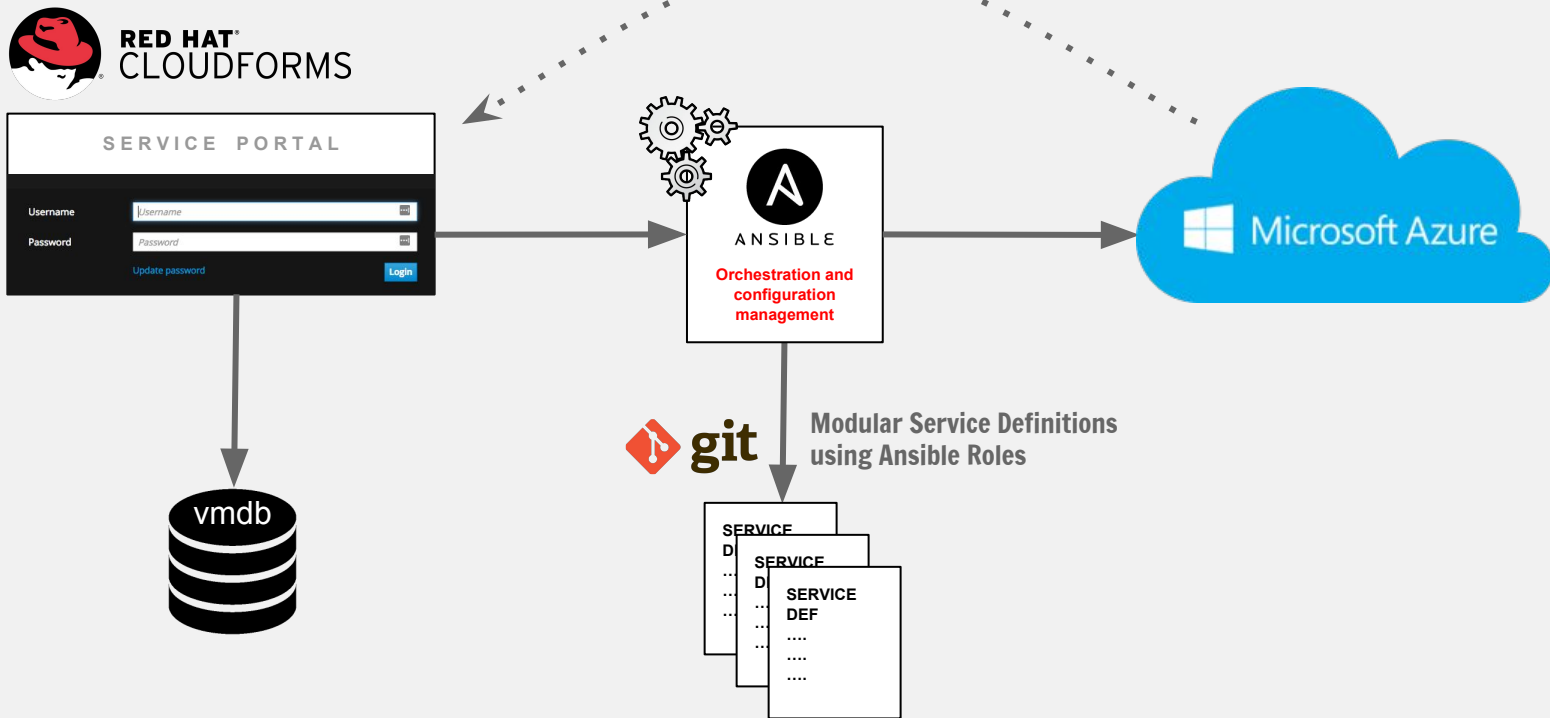


Microsoft Azure

Agenda

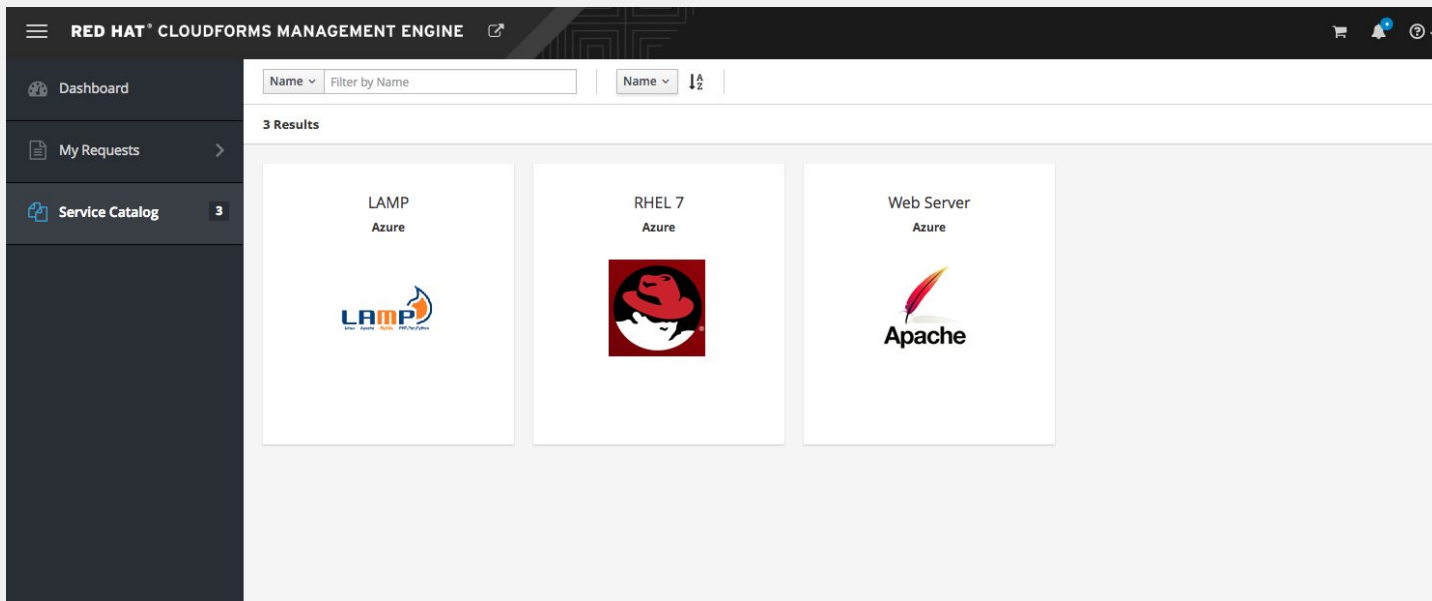
- Why Cloudforms and Ansible are great technologies to build a Service Catalog, Service Definition and Automation on top of public clouds. (15:00)
- Demo on how to configure Cloudforms and Tower to manage Azure (10:00)
- Anatomy of a playbook to provision workloads in Azure: (10:00)
- How to use tower as a centralized repository for our playbooks, managing access, logging, automation offerings, etc (10:00)
- Demonstration of how to create a new service offering in Cloudforms based on our Tower playbook. (10:00)
- Demonstration of how easy you can create multiple public Cloud offerings for your internal users, always based on the same easy to manage playbook structure. (10:00)
- Questions (15:00)

Building an internal public cloud offering with Cloudforms + Ansible Tower



Cloudforms self-service portal

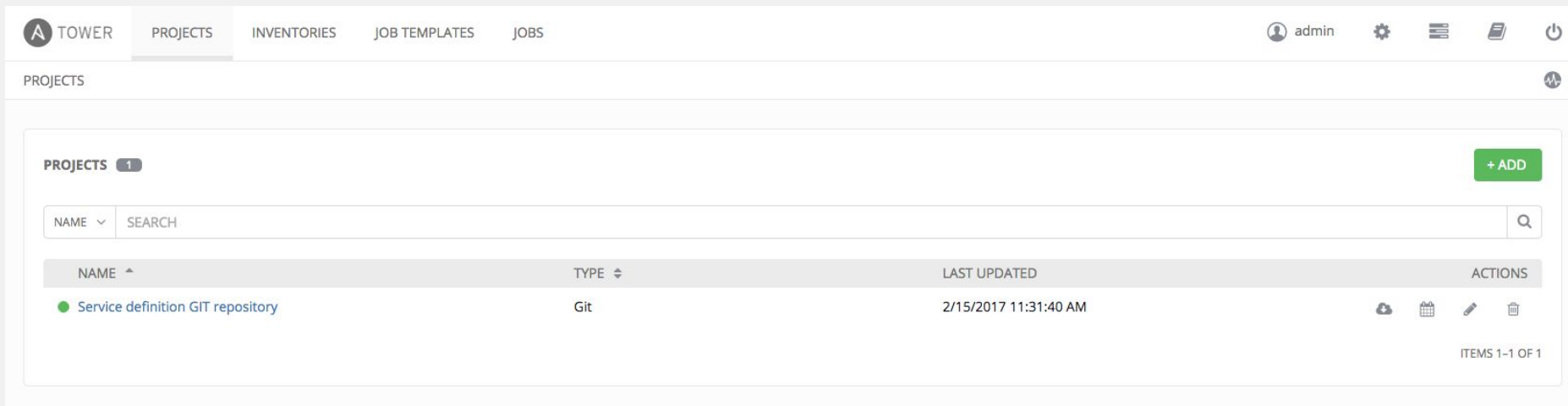
Offering a simplified self-service offering to your end-users







Cloudforms manages the access control and ownership of each service or VM.

Ansible Tower

Keep your service definition playbooks in a GIT repository



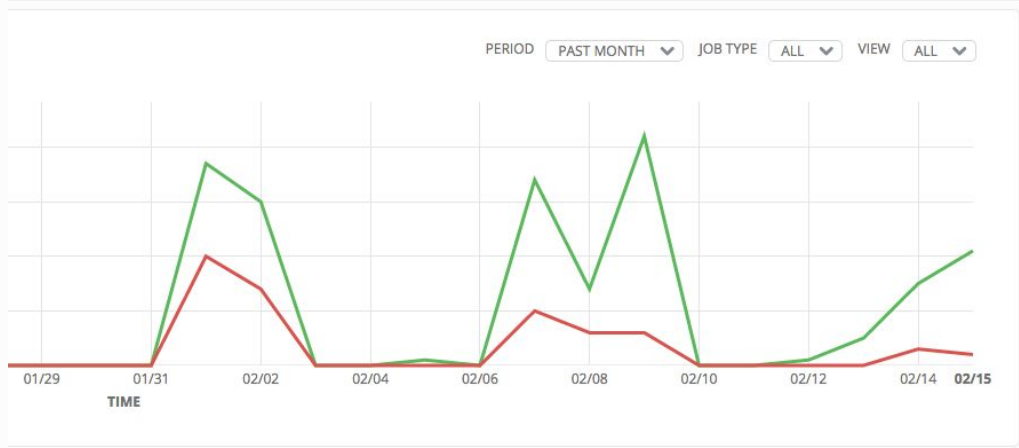
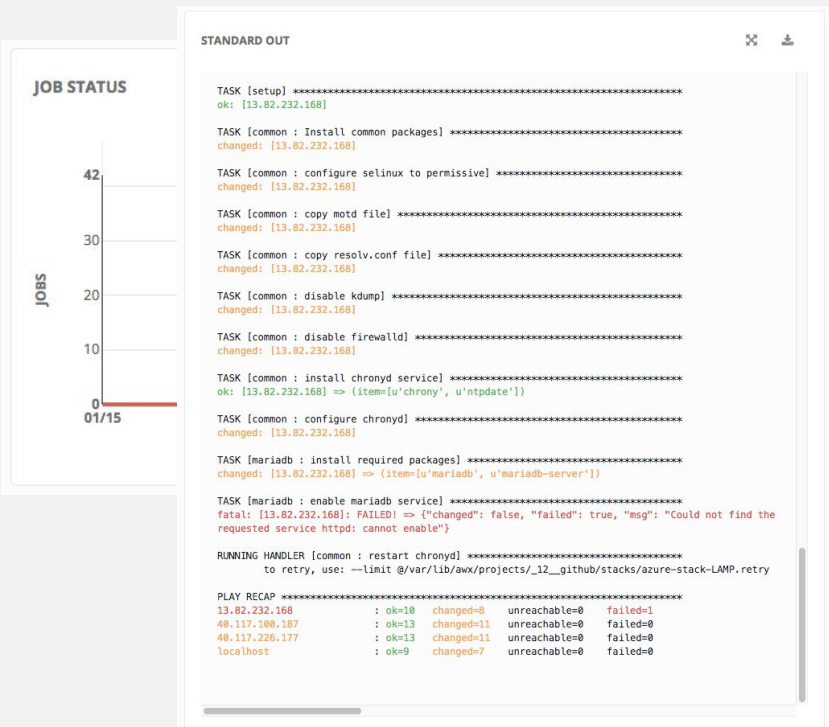
The screenshot displays the Ansible Tower web interface. At the top, there is a navigation bar with tabs for 'TOWER', 'PROJECTS', 'INVENTORIES', 'JOB TEMPLATES', and 'JOBS'. The 'PROJECTS' tab is currently selected. In the top right corner, there are icons for user profile (labeled 'admin'), settings, a menu, a document, and a power button. Below the navigation bar, the main content area is titled 'PROJECTS' and contains a sub-section 'PROJECTS 1' with a '+ ADD' button. A search bar with 'NAME' and 'SEARCH' fields is present. Below the search bar is a table with the following columns: 'NAME', 'TYPE', 'LAST UPDATED', and 'ACTIONS'. The table contains one entry: 'Service definition GIT repository' with type 'Git' and last updated '2/15/2017 11:31:40 AM'. The 'ACTIONS' column for this entry contains icons for refresh, calendar, edit, and delete. At the bottom right of the table area, it says 'ITEMS 1-1 OF 1'.

NAME ^	TYPE ↕	LAST UPDATED	ACTIONS
● Service definition GIT repository	Git	2/15/2017 11:31:40 AM	   

Easy revision control of your service definitions

Ansible Tower

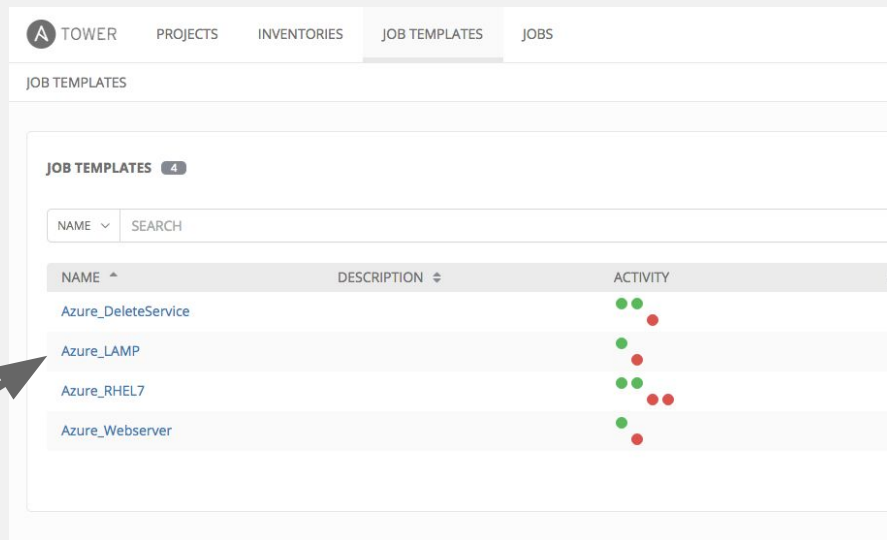
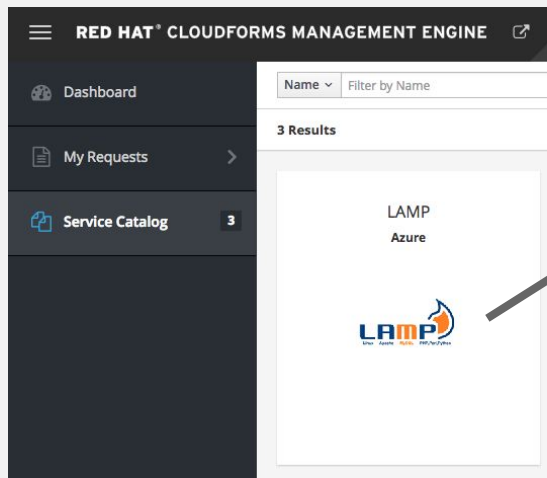
Monitoring service deployment successes and failures



- Understand what automation task failed immediately
- Understand automation patterns (successes / failures)
- Supervise in real-time any automation task

Cloudforms and Tower integration

Simply create a new service item in Cloudforms as a job template in Ansible Tower



Easily pass provisioning parameters from Cloudforms to Ansible Tower defining configuration settings for the playbook to apply

Configuring Azure as a provider in Cloudforms

Azure API

Here are the instructions from Microsoft to configure Azure for API access

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal>

Following this procedure will get you the following items:

- Application ID or Client ID : (same thing)
- Key :
- Directory ID or Tenant ID : (same thing)
- Subscription ID :

Configure Azure in Cloudforms (1/2)

In Cloudforms

Click on compute -> Clouds

Click on Configuration -> add a new cloud provider

Name : Azure

Type : Azure

Region : East US

Tenant ID : Directory ID

(On Azure, Azure Active Directory -> Properties)

Subscription ID : Azure Subscription ID

(On Azure, Subscriptions -> SubscriptionID)

Client ID : Azure Application ID

(On Azure, Azure Active Directory -> App Registration -> + ADD

Name : Cloudforms, Application type : Web App/ API, Signon URL:

<http://cf42.example.com> -> Create, then copy the Application ID)

Client key : Azure Key

(On Azure, Azure Active Directory -> App Registration -> your app -> Keys ->

description : Cloudforms, Expires : 1 year -> Save (the key will appear)

Do not click on validate yet !!

Configure Azure in Cloudforms (2/2)

Other things to do on Azure

1. Azure Active Directory -> Users and groups -> User setting -> App registration, turn it to ON
2. Subscriptions -> click on your subscription -> Access Control -> Add -> Select Role -> Contributor
3. Search for your application (Application id name) then select then ok

Back in Cloudforms

Click on validate



Cloud Intel >

Red Hat Insights >

Services >

Compute >

Configuration >

Networks >

Middleware >

Storage >

Control >

Automate >

Optimize >

Cloud Providers > Azure_RHUG3K (Summary) > Edit Cloud Providers 'Azure_RHUG3K'

Name	<input type="text" value="Azure_RHUG3K"/>
Type	<input type="text" value="Azure"/>
Region	<input type="text" value="East US"/>
Tenant ID	<input type="text" value="3af9bcc6-2f9d-43ae-a48b-ee552fec0e7e"/>
Subscription ID	<input type="text" value="1822d61e-bc55-4a21-ae0e-c1b7c00859e0"/>
Zone	<input type="text" value="default"/>

Credentials

Client ID	<input type="text" value="f5ece607-bdf8-4445-b0ec-2ceb63a63bd8"/>	
Client Key	<input type="text" value="....."/>	Change stored client key
	<input type="button" value="Validate"/>	

Configuring Azure In Ansible Tower

Ansible Tower Azure setup

Click on the gear -> credentials -> +ADD

Name : put a name

Type : Microsoft Azure Resource Manager

Subscription id : Azure Subscription ID

(On Azure, Subscriptions -> SubscriptionID)

Username : your azure username

Password: your azure password

Client id : Azure Application ID

(On Azure, Azure Active Directory -> App Registration -> + ADD

Name : Cloudforms, Application type : Web App/ API, Signon URL:

<http://cf42.example.com> -> Create, then copy the Application ID

Client secret : Keys

(On Azure, Azure Active Directory -> App Registration -> your app ->

Keys -> description : Cloudforms Expires : 1 year -> Save (the key will

appear)

Tenant ID : Azure Directory ID

(On Azure, Azure Active Directory -> Properties)

Click on Save

AZURE 

DETAILS

PERMISSIONS

*NAME

azure

DESCRIPTION

ORGANIZATION 

*TYPE 

Microsoft Azure Resource Manager

TYPE DETAILS

*SUBSCRIPTION ID 

1822d61e-bc55-4a21-ae0e-c1b7c00859e0

USERNAME

rhug_2017

PASSWORD

SHOW

CLIENT ID

9a0fa920-021c-4fc8-8c14-76d4ea715d07

CLIENT SECRET

SHOW

TENANT ID

3af9bcc6-2f9d-43ae-a48b-ee552fec0e7e

CANCEL

SAVE

CREDENTIALS 4

+ ADD

NAME  SEARCH



NAME 	DESCRIPTION 	TYPE	OWNERS	ACTIONS
azure		Microsoft Azure Resource Manager	admin	 

My reusable Azure Stacks playbook strategy

Provisioning and configuring a VM

Creating a resource group, provisioning VM(s), wait for SSH, configure those VM(s)

```
- hosts: localhost
  gather_facts: no
  roles:
    - { role: azure_service, service_name: "{{ service_id }}", location: "eastus", state: 'present' }
    - { role: azure_vm, service_name: "{{ service_id }}", server_group: 'group1', server_qty: "{{ group1_qty }}" }
    - { role: sshwait, server_group: "{{ hostvars[inventory_hostname].groups.group1 }}" }

- hosts: group1
  become: true
  roles:
    - { role: common }
```

What if I want that VM to be a Web server?

Just adding an “apache” role to my group1, with a “giturl” variable to pull the content from...

```
- hosts: localhost
gather_facts: no
roles:
  - { role: azure_service, service_name: "{{ service_id }}", location: "eastus", state: 'present' }
  - { role: azure_vm, service_name: "{{ service_id }}", server_group: 'group1', server_qty: "{{ g
  - { role: sshwait, server_group: "{{ hostvars[inventory_hostname].groups.group1 }}" }

- hosts: group1
become: true
roles:
  - { role: common }
  - { role: apache, gitrepo: "{{ giturl }}" }
```

What if I need database servers as well?

In the same modular approach, I can provision a second group of servers (group2), which will have the “mariadb” role attached to it instead of the “apache”.

Lots of roles like apache or mariadb available on <https://galaxy.ansible.com/>

```
- hosts: localhost
gather_facts: no
roles:
  - { role: azure_service, service_name: "{{ service_id }}", location: "eastus", s
  - { role: azure_vm, service_name: "{{ service_id }}", server_group: 'group1', se
  - { role: azure_vm, service_name: "{{ service_id }}", server_group: 'group2', se
  - { role: sshwait, server_group: "{{ hostvars[inventory_hostname].groups.group1
  - { role: sshwait, server_group: "{{ hostvars[inventory_hostname].groups.group2

- hosts: group1
become: true
roles:
  - { role: common }
  - { role: apache, gitrepo: "{{ giturl }}" }

- hosts: group2
become: true
roles:
  - { role: common }
  - { role: mariadb }
```

Using variables to pass configuration details

Each playbook is expecting input variables, which can be defined in my playbook, in Tower or even as a parameter from Cloudforms self-service UI.

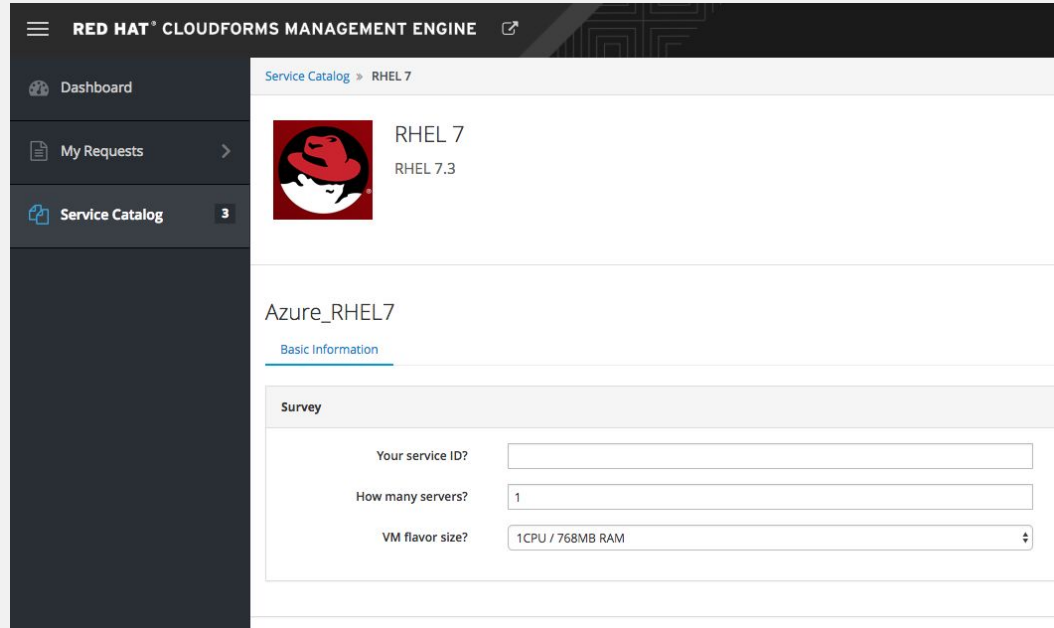
```
# INPUT VARIABLES:
# - service_id: unique service name identifier
# - group1_qty: number of VMs
# - group1_image: Azure image description
  # {{ group1_image }}:
  # offer: RHEL
  # publisher: RedHat
  # sku: '7.3'
  # version: latest
# - group1_vmsize: Azure image size
# - group1_ports: Azure network open ports in security group
# - giturl: Web content GIT URL
```

Using variables to pass configuration details

In Cloudforms, some of those variables can be replaced by a user-friendly question.

Instead of showing a list of flavor sizes variables like “Basic_A0, Basic_A1, Basic_A2”, I am showing the exact amount of CPU and RAM you’d get.

That said, Cloudforms will pass the right Basic_Ax value to Tower, based on the option picked by the end-user.



My playbook examples can be found here:

https://github.com/rhug2017/ansible_azure



THANK YOU



plus.google.com/+RedHat



facebook.com/redhatinc



linkedin.com/company/red-hat



twitter.com/RedHatNews



youtube.com/user/RedHatVideos