Robocode on Cloud Foundry

Team A1

Karan Shukla
John Southerland
Tom Hill
Romi Padam
Avisha Manik

Table of Contents

Team A Final Report

➢ Study the PaaS platforms

Google App Engine

The Google App Engine (GAE) is now available in eight regions, located in: US, Europe and Asia/Pac. GAE provides a platform of services, PaaS, supporting most popular languages including: Node.js, Java, Ruby, C#, Go, Python, and PHP. Additionally, GAE supports the ability to bring your own custom stack via a Docker image. The managed services (APIs) list continues to grow with the following general categories offered: SQL, NoSQL, monitoring/diagnostics/management tools, cloud pub/sub, big data, IoT, machine learning, and a complete developer tools suite. The service level agreement is built on availability and a future credit system only. No SLA provisions are made for additional performance requirements.

Cloud Foundry

Cloud Foundry can be installed onto any infrastructure

- Deployment is handled by Bosh
  - Bosh-Lite is used to deploy Cloud Foundry on local machines - Bosh can also be integrated with platforms such as AWS and Azure
  - Bosh can deploy both VMs and containers (they are both referred to as “instances” in Bosh)
  - The “Director” handles orchestration of Bosh instances. Each Bosh instance contains an “Agent” that communicates with the Director via NATS pub-sub messaging.
    i. This is similar to the Master-Slave paradigm seen in other platforms
    ii. The Director is created using bosh-init
  - Bosh’s deployment manifest is a YAML file, and can handle deployment, network, and resource configuration. This is done with the bosh create-env command.
- Architecture is known as “Diego”
  - The “Brain” or “Auctioneer” handles fault-tolerance
    i. It classifies jobs into “tasks” (run-once, finite) and “long-running processes”, or “LRPs” (run continuously, indefinite)
    ii. The Auctioneer prioritizes creating one instance of each LRP first. Afterwards, it runs all tasks.
    iii. Once all tasks have begin running, it distributes remaining instances of the LRPs broadly over multiple Availability Zones and Cells, prioritizing those Cells with the lightest load.
  - Worker nodes are known as “Cells”
  - The “Loggregator” is a service that aggregates logs across users and applications (“logs” are anything that is written to stderr or stdout)
    i. It does this by communicating with “Metron Agent”, the part of a cell that gathers and forwards logs to the Loggregator
  - The Cloud Controller handles app deployment, cooperating with the Brain
  - The router distributes incoming network traffic

- Most actions, such as those relating to authorization, configuration, and deployment, are handled from the Cloud Foundry CLI
    - Use cf set-org-role and cf set-space-role to assign roles to users
    - Use cf push to deploy. Manifest YAML files or command-line arguments can be used to set parameters (including application path, buildpacks, disk quota, memory limit, and number of instances).
    - Use cf scale to increase the number of instances or the allocated disk space or memory for an application
    - Use cf map-route to map a route to a deployed application. Alternatively, the cf push <appname> -d <domain name> --hostname <hostname> can be used to simultaneously deploy an app and map a route to it.
    - Use cf set-env to set environmental variables to configure applications
    - Use cf logs to read Loggregator
- Cloud Foundry supports Java, Node.js, Ruby, Go, PHP, and Python
    - Application dependencies are distributed via buildpacks
    - Java developers can deploy applications to Cloud Foundry instances directly from Eclipse
- Role-Based Access Control via User Account and Authentication (UAA)
    - Org - a pool of resources, applications, and domains shared by a group of users
    - Space - a shared location for app development, deployment, and maintenance
    - Administrators cannot create their own roles - there are a fixed selection of roles
        i. Admin - has all permissions
        ii. Admin Read-Only - can read everything
        iii. Global Auditor - can read everything, except “secrets” (such as environmental variables)
        iv. Org/Space Managers - have all permissions within an org/space
        v. Org/Space Auditors - can read an org/space but cannot edit it

RackHD

RackHD sits on top of physical hardware, and is part of the “M&O” (management and orchestration) stack.

- Features include:
    - Automatically discovering and cataloging system resources
    - Configuring hardware, ports, and network
    - Logging
    - Fault detection
    - Provide analytics about hardware performance or failure
    - Automated installation of operating systems and software
- Provides a live REST API (with JSON output) to access the above features, allowing for automated hardware management
- Where does RackHD fit into the stack?
    - CloudFoundry/Heroku are application-deployment automation services built on top of infrastructure layers like AWS and Openstack.
    - AWS and Openstack are built on infrastructure orchestration+management tools like Ansible
    - RackHD basically combines the AWS-like and Ansible-like layers into a single REST API layer that can be accessed by Cloud Foundry and Heroku.

- Deployment
    - RackHD can be configured to work on a single network or on several network.
    - Prerequisites
        i. Configuring the network interface controllers
        ii. node.js
        iii. Erlang
        iv. RabbitMQ
        v. MongoDB
        vi. snmp
        vii. ipmitool
        viii. ansible
        ix. amtterm
        x. dhcp
    - Then, configuration needs to establish security certificates and network endpoints
    - After that, install bosh. Then, install Cloud Foundry.

➢ Deploy Cloud Foundry and Services

We deployed Cloud Foundry on top of BOSH-Lite on node 1 and the MySQL database on node 8. We can connect Cloud Foundry organizations to the MySQL database via security groups. We have two applications we can run on it via `cf push`, these applications being cf-helloworld and Robocode.

➢ Multitenancy Access Control

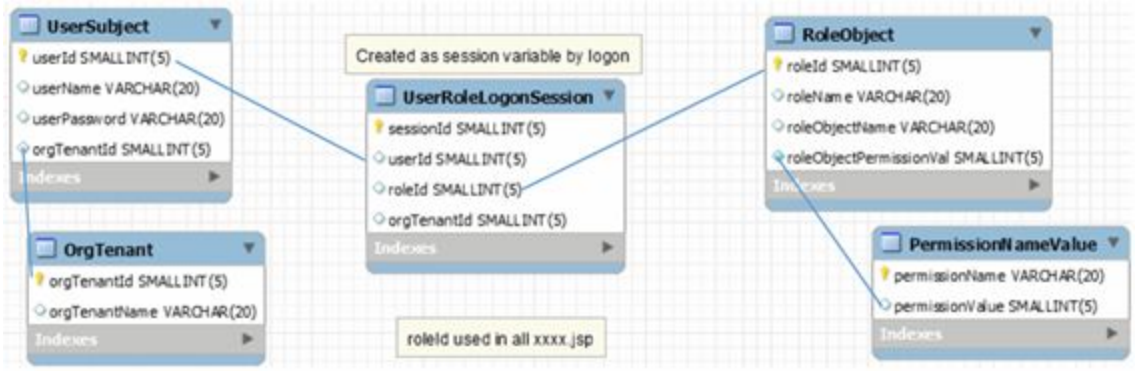
We have Cloud Foundry's UAA service deployed and running. UAA handles organizations, roles, and users. Applications running on our Cloud Foundry instance can use our UAA service to determine a user's role; based on this, the application may provide different rights to the user. For Robocode, we have created a login screen that uses UAA's user authentication service as a backend. This login is required to access the rest of the Robocode application.

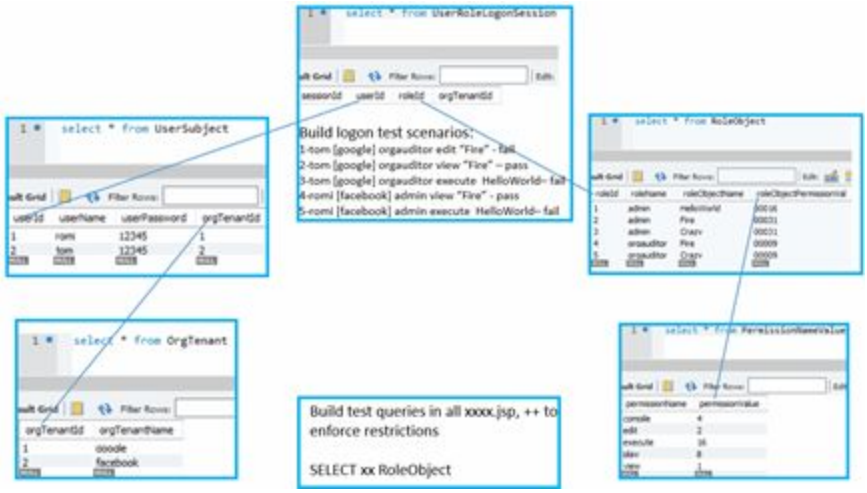
We have an admin user who has Admin privileges in Cloud Foundry, granting him or her the right to create new tenants (orgs) and users, and to assign roles to users. We can also create Org Managers who serve as the tenant admins. They are able to create users within their tenant and assign roles to those users. We have a role hierarchy, where Org Manager is parent to Org User and Space Manager, Space Manager is parent to Space Developer, and Admin is parent to Org Manager. We use an API composed of the multitenant role-based UAA service.

We chose to adopt RBAC-1. Users can have multiple roles, as this is a necessity as part of the Cloud Foundry application and reflects that users in our domain may belong to multiple tenants. Furthermore, we have adopted a Role Hierarchy based on organizations and spaces. We did not choose to use RBAC-2, which would add Conflicts-of-Interests, because this would make it impossible to have an Admin user who oversees all operations in our Cloud Foundry instance. Thus, we have satisfied all access control requirements set forth within the final submission requirements.

➢ **Access Control design alternatives considered:**

- 1. Traditional additions to robocode data model and implementation in MySQL





- 2. Extension to the existing UAA datastore to include the following elements:
  - Organization-tenant identification
  - User identification
  - Password
  - Role1, Role2, Role3…

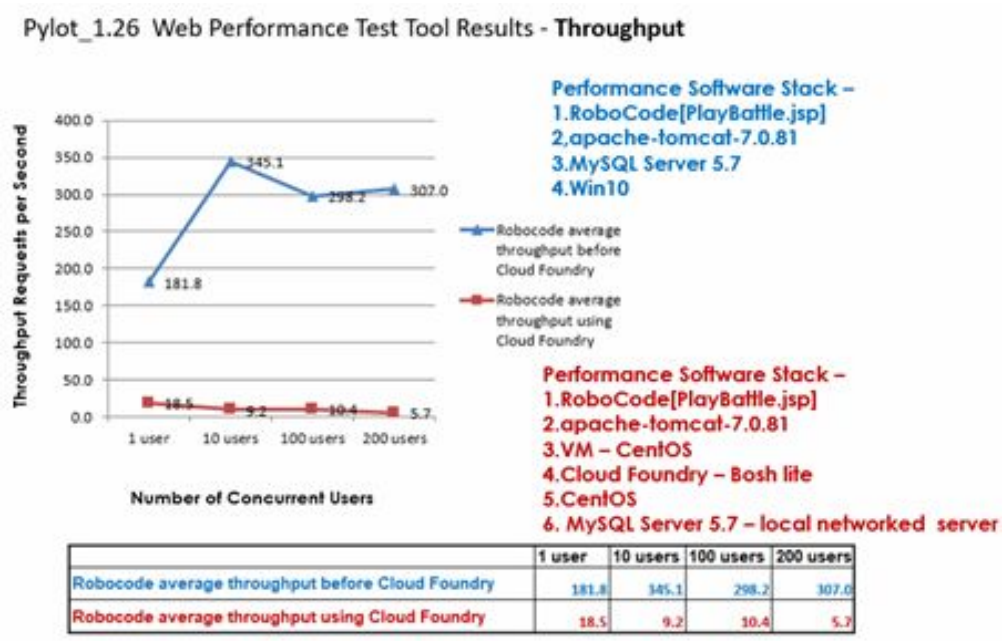
  This alternative eliminates the data tables required to keep user, user/role, role/object and permissions on the robocode database. It also eliminates the need to insert new database accesses in the robocode java server pages to enforce access restrictions. The robocode java server pages can access the logged-on access restriction information as session variables

➢ **Demo**

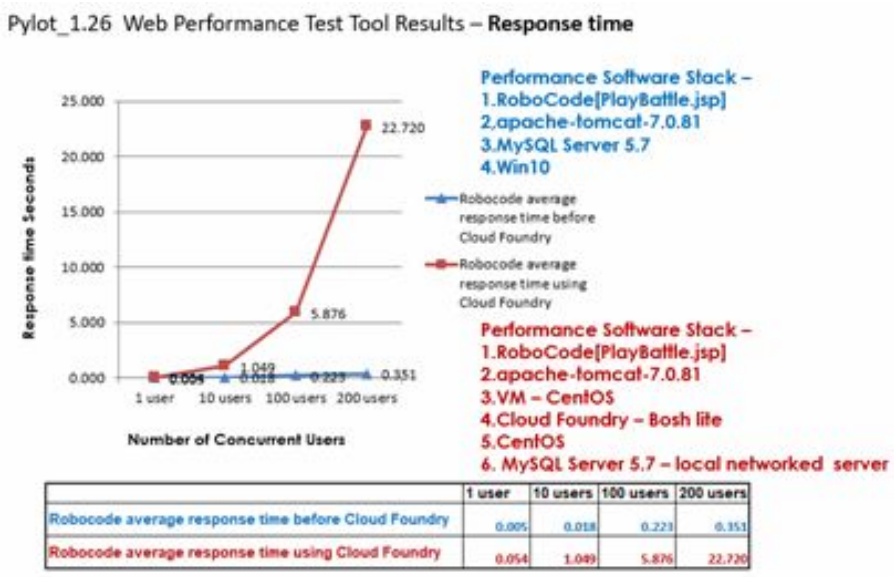
- Create tenants: can be done with `cf create-org <orgname>`
- Create users: can be done with `cf create-user` followed by `cf set-org-role <user> <org> <role>`
- Dynamically change access control policy: `cf set-org-role` and `cf unset-org-role <user> <org> <role>`
- Demonstrate access control policy is enforced: Login to Robocode - you will only be able to perform actions that you have permission for.

➢ **Performance Study**

Robocode before Cloud Foundry and Lab Cloud Foundry Implementation performance using Pylot.



	1 user	10 users	100 users	200 users
Robocode average throughput before Cloud Foundry	181.8	345.1	298.2	307.0
Robocode average throughput using Cloud Foundry	18.5	9.2	10.4	5.7



	1 user	10 users	100 users	200 users
Robocode average response time before Cloud Foundry	0.005	0.018	0.223	0.351
Robocode average response time using Cloud Foundry	0.054	1.049	5.876	22.720

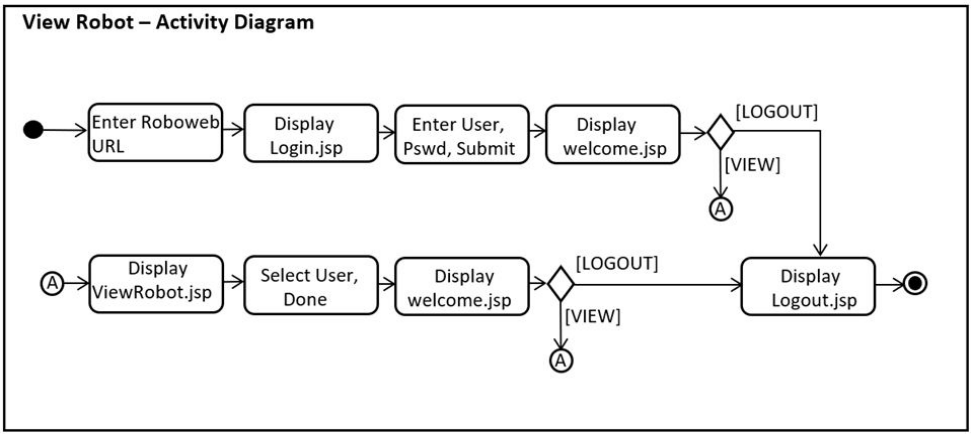
➢ User Manual

Go to https://roboweb.bosh-lite.com/roboweb, login, and play robocode.

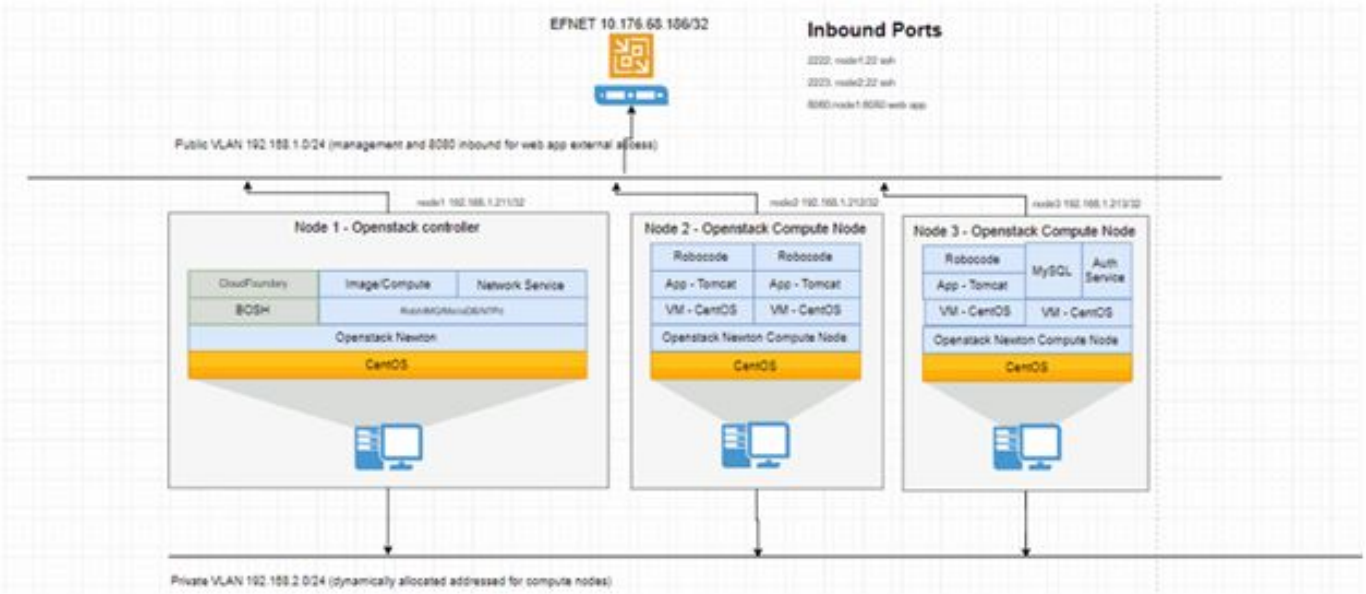
Visit https://github.com/jbsouthe/cloudComputing/wiki for more details on the system's functionality and the creation process behind the project.

➢ System Architecture

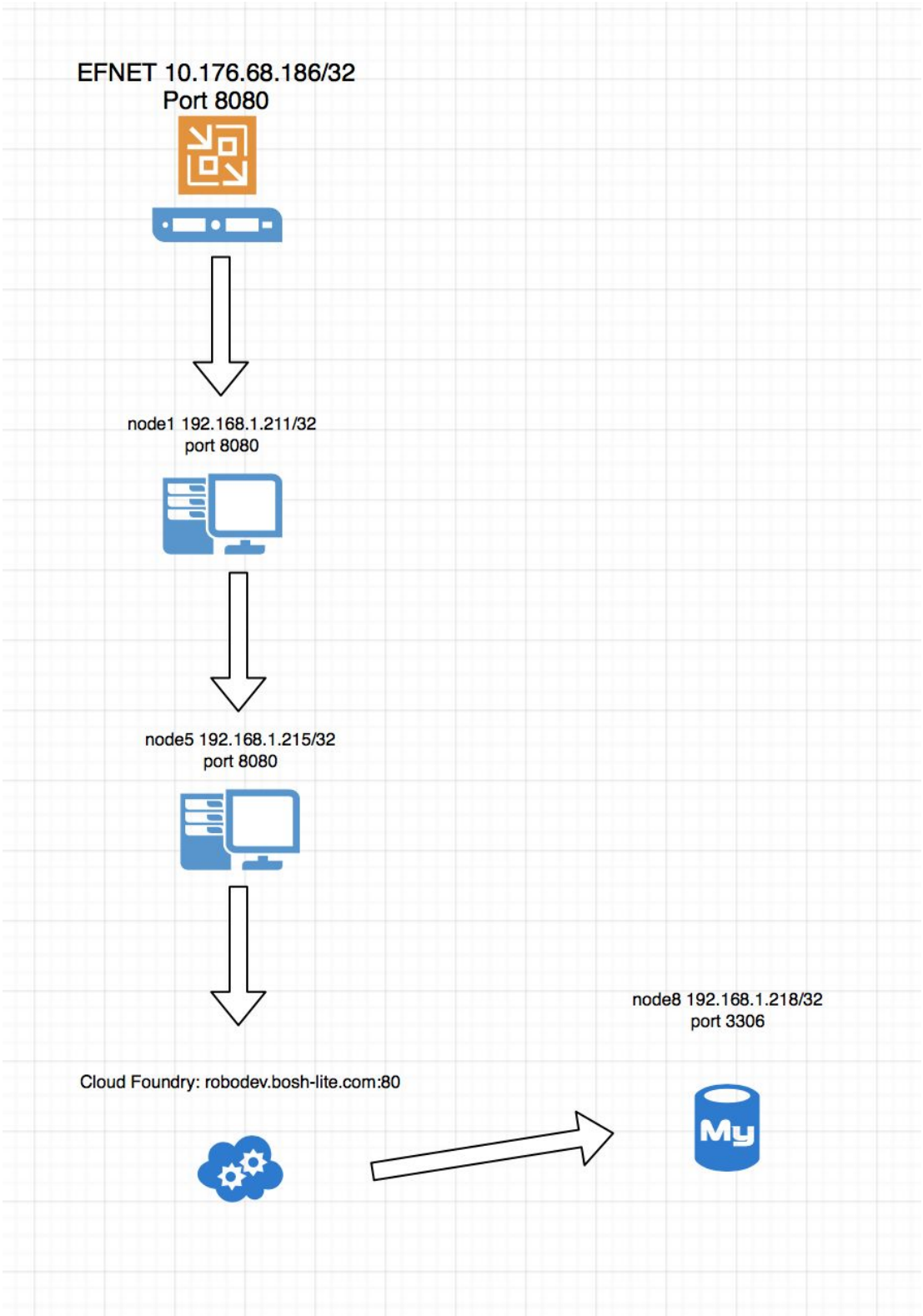
➢ Activity Workflow



➢ **Cluster Architecture**



➢ **Network Diagram**



APPENDIX A -

Pylot_1.26 Web Performance Test Tool Experiments Detailed Results

1- Before Cloud Foundry

Pylot - robo1 Performance Results

report generated: 11/22/2017 14:49:41
test start: 11/22/2017 14:48:41
test finish: 11/22/2017 14:49:41

Workload Model

test duration (secs)	60
agents	1
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	11088
errors	0
data received (bytes)	113962784

Response Time (secs)		Throughput (req/sec)	
avg	0.005	avg	181.770
stdev	0.004	stdev	51.928
min	0.003	min	16
50th %	0.004	50th %	207
80th %	0.005	80th %	221
90th %	0.007	90th %	227
95th %	0.009	95th %	228
99th %	0.013	99th %	240
max	0.394	max	240

Pylot - robot10 Performance Results

report generated: 11/22/2017 15:01:12
test start: 11/22/2017 15:00:06
test finish: 11/22/2017 15:01:13

Workload Model

test duration (secs)	67
agents	10
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	23120
errors	0
data received (bytes)	237447097

Response Time (secs)		Throughput (req/sec)	
avg	0.018	avg	345.075
stdev	0.007	stdev	74.887
min	0.005	min	9
50th %	0.017	50th %	364
80th %	0.021	80th %	382
90th %	0.024	90th %	388
95th %	0.029	95th %	397
99th %	0.050	99th %	415
max	0.165	max	415

Pylot - robot100 Performance Results

report generated: 11/23/2017 09:50:55
test start: 11/23/2017 09:49:32
test finish: 11/23/2017 09:50:08

Workload Model

test duration (secs)	36
agents	100
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	11033
errors	0
data received (bytes)	113569146

Response Time (secs)		Throughput (req/sec)	
avg	0.223	avg	298.189
stdev	0.090	stdev	65.348
min	0.035	min	17
50th %	0.226	50th %	311
80th %	0.248	80th %	337
90th %	0.269	90th %	341
95th %	0.284	95th %	342
99th %	0.415	99th %	345
max	1.295	max	345

Pylot - robot200 Performance Results

report generated: 11/23/2017 10:04:42
test start: 11/23/2017 10:03:31
test finish: 11/23/2017 10:03:52

Workload Model

test duration (secs)	21
agents	200
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	6447
errors	0
data received (bytes)	66162592

Response Time (secs)		Throughput (req/sec)	
avg	0.351	avg	307.000
stdev	0.181	stdev	32.120
min	0.007	min	256
50th %	0.351	50th %	299
80th %	0.527	80th %	325
90th %	0.548	90th %	364
95th %	0.581	95th %	370
99th %	0.693	99th %	376
max	0.823	max	376

2- Using Cloud Foundry

Pylot - cf1 Performance Results

report generated: 12/09/2017 13:13:36
test start: 12/09/2017 13:12:25
test finish: 12/09/2017 13:13:20

Workload Model

test duration (secs)	55
agents	1
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	1014
errors	0
data received (bytes)	11524110

Response Time (secs)		Throughput (req/sec)	
avg	0.054	avg	18.436
stdev	0.608	stdev	23.730
min	0.011	min	1
50th %	0.015	50th %	1
80th %	0.017	80th %	52
90th %	0.020	90th %	56
95th %	0.024	95th %	63
99th %	0.039	99th %	64
max	15.056	max	64

Pylot - cf10 Performance Results

report generated: 12/09/2017 13:19:06
test start: 12/09/2017 13:18:02
test finish: 12/09/2017 13:18:36

Workload Model

test duration (secs)	34
agents	10
rampup (secs)	0
interval (millisecs)	0

Results Summary

requests	314
errors	0
data received (bytes)	3568610

Response Time (secs)		Throughput (req/sec)	
avg	1.049	avg	9.235
stdev	5.497	stdev	25.733
min	0.015	min	1
50th %	0.041	50th %	1
80th %	0.071	80th %	1
90th %	0.113	90th %	13
95th %	0.207	95th %	91
99th %	31.168	99th %	96
max	31.831	max	96

Pylot - cf100-5sec Performance Results

report generated: 12/09/2017 13:25:48
test start: 12/09/2017 13:24:33
test finish: 12/09/2017 13:25:48

Workload Model

test duration (secs)	75
agents	100
rampup (secs)	0
interval (millisecs)	5000

Results Summary

requests	776
errors	4
data received (bytes)	8773780

Response Time (secs)		Throughput (req/sec)	
avg	5.876	avg	10.347
stdev	9.742	stdev	19.044
min	0.012	min	1
50th %	3.022	50th %	1
80th %	7.062	80th %	21
90th %	15.096	90th %	34
95th %	31.096	95th %	65
99th %	31.300	99th %	81
max	62.896	max	81

Pylot - cf200-5sec Performance Results

report generated: 12/09/2017 13:28:28
test start: 12/09/2017 13:26:46
test finish: 12/09/2017 13:28:28

Workload Model

test duration (secs)	102
agents	200
rampup (secs)	0
interval (millisecs)	5000

Results Summary

requests	581
errors	141
data received (bytes)	5011965

Response Time (secs)		Throughput (req/sec)	
avg	22.725	avg	5.696
stdev	24.593	stdev	11.978
min	0.012	min	1
50th %	15.075	50th %	1
80th %	60.019	80th %	3
90th %	62.814	90th %	17
95th %	63.237	95th %	34
99th %	70.039	99th %	43
max	70.984	max	72

APPENDIX B -

Additional Database Research Results

A complete Redis installation and test script is provided below:

1. Redis in-memory datastore installed on Centos 7 development environment.

2. Install Redis datastore service on Lab node8 side-by-side with MySQL.

3. Build the complete ‘ROBOT” table as a sorted set in Redis on node8.

4. Build Redis install script for node8.

5. Show Redis-server process view-ps ax.

```
8357 ?        S      0:00 [kworker/2:2]
8385 pts/0    S      0:00 su root
8393 pts/0    S      0:00 bash
8434 ?        S      0:00 [kworker/1:2]
8451 ?        S      0:00 sleep 60
8463 ?        Ssl    0:00 /usr/bin/redis-server 127.0.0.1:6379
```

6. Construct first test database table for “PlayRobot” transaction to be replaced by Redis.

```
SELECT * from robot
id userID        packageID     robotID
1  User          sample        Crazy
2  User          sample        Corners
3  User          sample        Fire
4  User          sample        Interactive
```

7. Add test database table to Redis as a sorted set.

```
127.0.0.1:6379> quit
[root@osboxes Downloads]# cat roboload.txt | xargs  -L1
redis-cli
(integer) 1
(integer) 1
(integer) 1
(integer) 1

[root@osboxes Downloads]# redis-cli
127.0.0.1:6379> scan 0
1) "0"
2) 1) "robot"
127.0.0.1:6379> ZRANGE robot 0 10 WITHSCORES
1) "User|sample|Crazy"
2) "1"
3) "User|sample|Corners"
4) "2"
5) "User|sample|Fire"
6) "3"
7) "User|sample|Interactive"
8) "4"
127.0.0.1:6379> quit
[root@osboxes Downloads]#
```

8. Node8 Redis server information shown.

9. Node8 Redis server benchmark results shown.

10. Node8 Python program written to test Redis execution environment setup.