

Cloud Computing

Analysis of PaaS and SaaS

Group C1

Team members

Dhruv Sangvikar (dgs160230)

Sabarish Nadarajan (sxn164530)

Arun Prakash (axt161330)

Aayush Bhandari (axb167830)

1. Introduction	2
1.1 Goal Of The Project	2
1.1 Study of Related Work	2
2. CloudFoundry - Detailed Study	5
2.1 Architecture	5
2.2 BOSH	8
2.3 STEMCELL	10
3. User Account and Authentication	11
3.1 Commands to register the client	13
3.2 Scopes	14
Cloud Controller Scopes	14
UAA Scopes	15
3.3 Resource Server Information Retrieval	15
Username Retrieval	15
Space/Org Information Retrieval	15
Role Information Retrieval	15
API Endpoints	15
4. Access Control Mechanism	16
4.1 Access Control Model for Robocode on Cloud Foundry	17
4.1.1 Access Control Model within each tenant	17
4.1.2 Cross Tenant Access Control	19
4.2 Commands for Access Control	20
4.3 Implementation of Access Control	21
5. Database Schema	26
5.1 Schema	26
6. Performance Analysis	28
Proposed performance analysis	28
References	30

1. Introduction

1.1 Goal Of The Project

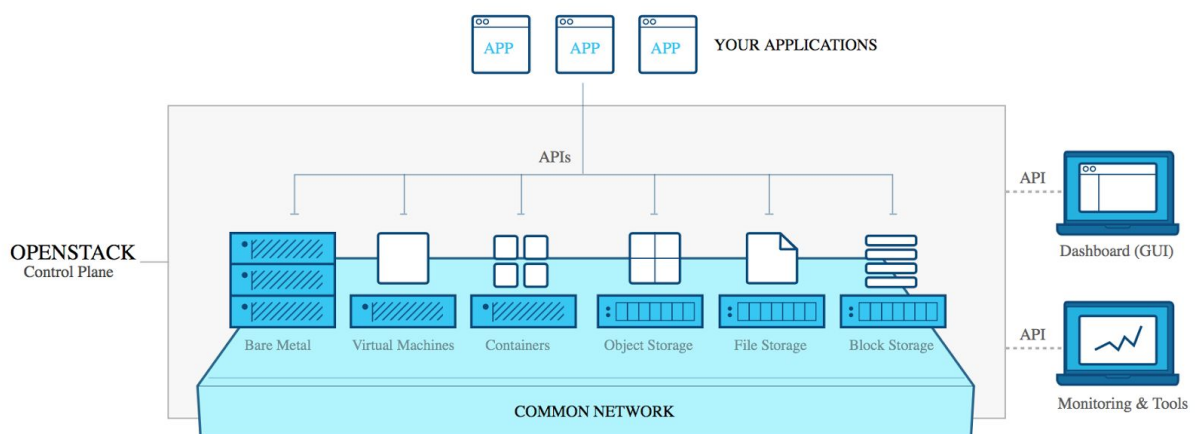
Our goal is to deploy Robocode, which is a web application, on Cloud Foundry. We try to achieve the following

- Authentication and Authorization mechanism for secured access to application and resources
- Create Architecture for Access Control Policy Enforcements within Cloud Foundry and mapping to the robocode application
- Design database schemas to enforce data restrictions such as read/write to users that are using the application

1.1 Study of Related Work

Open Stack:

Open source software for creating private and public clouds. OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure.



OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. The following are OpenStack services.

Compute Services**NOVA** - Compute Service**GLANCE** - Image Service**IRONIC** - Bare Metal Provisioning Service**MAGNUM** - Container Orchestration Engine

Provisioning

STORLETS - Computable Object Store**ZUN** - Container Management Service**Storage, Backup & Recovery****SWIFT** - Object Store**CINDER** - Block Storage**MANILA** - Shared Filesystems**KARBOR** - Application Data Protection as a Service**FREEZER** - Backup, Restore, and Disaster Recovery**Networking & Content Delivery****NEUTRON** - Networking**DESIGNATE** - DNS Service**DRAGONFLOW** - Neutron Plugin**KURYR** - Container plugin**OCTAVIA** - Load Balancer**TACKER** - NFV Orchestration**TRICIRCLE** - Networking Automation for Multi-Region Deployments**Data & Analytics****TROVE** - Database as a Service**SAHARA** - Big Data Processing Framework

Provisioning

SEARCHLIGHT - Indexing and Search**Security, Identity & Compliance****KEYSTONE** - Identity service**BARBICAN** - Key Management**CONGRESS** - Governance**MISTRAL** - Workflow service**Management Tools****HORIZON** - Dashboard**OPENSTACK CLIENT (CLI)** - Command-line client**RALLY** - Benchmark service**SENLIN** - Clustering service**VITRAGE** - RCA (Root Cause Analysis service)**WATCHER** - Optimization Service**Deployment Tools****CHEF OPENSTACK** - Chef cookbooks for OpenStack**KOLLA** - Container deployment**OPENSTACK CHARMS** - Juju Charms for OpenStack**OPENSTACK-ANSIBLE** - Ansible Playbooks for OpenStack**PUPPET OPENSTACK** - Puppet Modules for OpenStack**TRIPLEO** - Deployment service**Application Services****HEAT** - Orchestration**ZAQAR** - Messaging Service**MURANO** - Application Catalog**SOLUM** - Software Development Lifecycle Automation**Monitoring & Metering****CEILOMETER** - Metering & Data Collection Service**CLOUDKITTY** - Billing and chargebacks**MONASCA** - Monitoring**AODH** - Alarming Service**PANKO** - Event, Metadata Indexing Service**Metal as a Service (MAAS)**

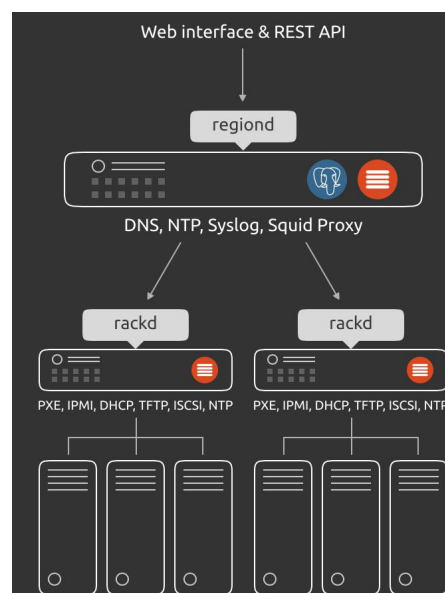
MAAS is a server provisioning tool which offers complete automation of your physical servers for amazing data centre operational efficiency. On premises, open source and supported.

Features

- Dynamic hardware resource management with Intel RSD
- Manage the machine's storage with RAID, bcache, LVM and more
- Full chassis convergence, including Cisco UCS, HP Moonshot and more
- Configure the machine's network interfaces with bridges, VLANs, bonds and more
- Observes and catalogs every IP address on the network (IPAM)
- Monitors and tracks critical services to ensure proper operations.

- Built-in highly available DHCP and DNS.
- REST API, Web UI and CLI

MAAS is mission critical service, providing infrastructure coordination upon which HPC and cloud infrastructures depend. High availability in the region controller is achieved at the database level. The region controller will automatically switch gateways to ensure high availability of services to network segments in the event of a rackd failure.



MAAS has a tiered architecture with a central postgres database backing a 'Region Controller (regiond)' that deals with operator requests. Distributed Rack Controllers (rackd) provide high-bandwidth services to multiple racks. The controller itself is stateless and horizontally scalable, presenting only a REST API.

Rack Controller (rackd) provides DHCP, IPMI, PXE, TFTP and other local services. They cache large items like operating system install images at the rack level for performance but maintain no exclusive state other than credentials to talk to the controller.

2. CloudFoundry - Detailed Study

Cloud Foundry is an open-source platform as a service (PaaS) that provides you with a choice of clouds, developer frameworks, and application services. Cloud Foundry is designed to be configured, deployed, managed, scaled, and upgraded on any cloud IaaS provider. Cloud Foundry achieves this by leveraging BOSH, an open source tool for release engineering, deployment, lifecycle management, and distributed systems monitoring.

2.1 Architecture

Cloud Foundry components include a self-service application execution engine, an automation engine for application deployment and lifecycle management, and a scriptable command line interface (CLI), as well as integration with development tools to ease deployment processes. Cloud Foundry has an open architecture that includes a buildpack mechanism for adding frameworks, an application services interface, and a cloud provider interface.

[8]

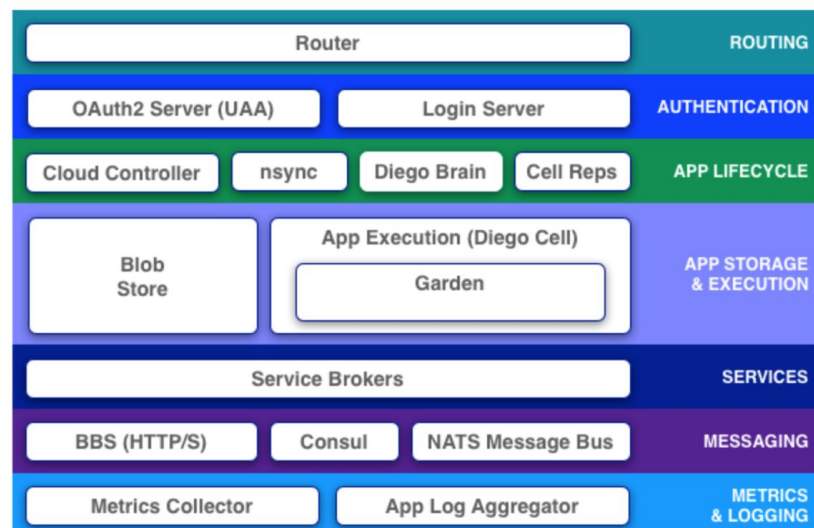


Figure 2.1: Cloud Foundry Architecture

Routing Router

The router routes incoming traffic to the appropriate component, either a Cloud Controller component or a hosted application running on a Diego Cell.

The router periodically queries the Diego Bulletin Board System (BBS) to determine which cells and containers each application currently runs on. Using this information, the router recomputes new routing tables based on the IP addresses of each cell virtual machine (VM) and the host-side port numbers for the cell's containers.

The Gorouter routes traffic coming into Cloud Foundry to the appropriate component, whether it is an operator addressing the Cloud Controller or an application user accessing an app running on a Diego Cell. The router is implemented in Go. Implementing a custom router in Go gives the router full control over every connection, which simplifies support for WebSockets and other types of traffic (for example, through HTTP CONNECT). A single process contains all routing logic, removing unnecessary latency.

Authentication

OAuth2 Server (UAA) and Login Server

The OAuth2 server (the UAA) and Login Server work together to provide identity management.

The primary role of the UAA is as an OAuth2 provider, issuing tokens for client apps to use when they act on behalf of Cloud Foundry users. In collaboration with the login server, the UAA can authenticate users with their Cloud Foundry credentials, and can act as an SSO service using those, or other, credentials.

The UAA has endpoints for managing user accounts and for registering OAuth2 clients, as well as various other management functions.

App Lifecycle

Cloud Controller and Diego Brain

The Cloud Controller (CC) directs the deployment of applications. To push an app to Cloud Foundry, you target the Cloud Controller. The Cloud Controller then directs the Diego Brain through the CC-Bridge components to coordinate individual Diego cells to stage and run applications.

The Cloud Controller provides REST API endpoints for clients to access the system. The Cloud Controller maintains a database with tables for orgs, spaces, services, user roles, and more.

Refer to the following diagram for information about internal and external communications of the Cloud Controller.

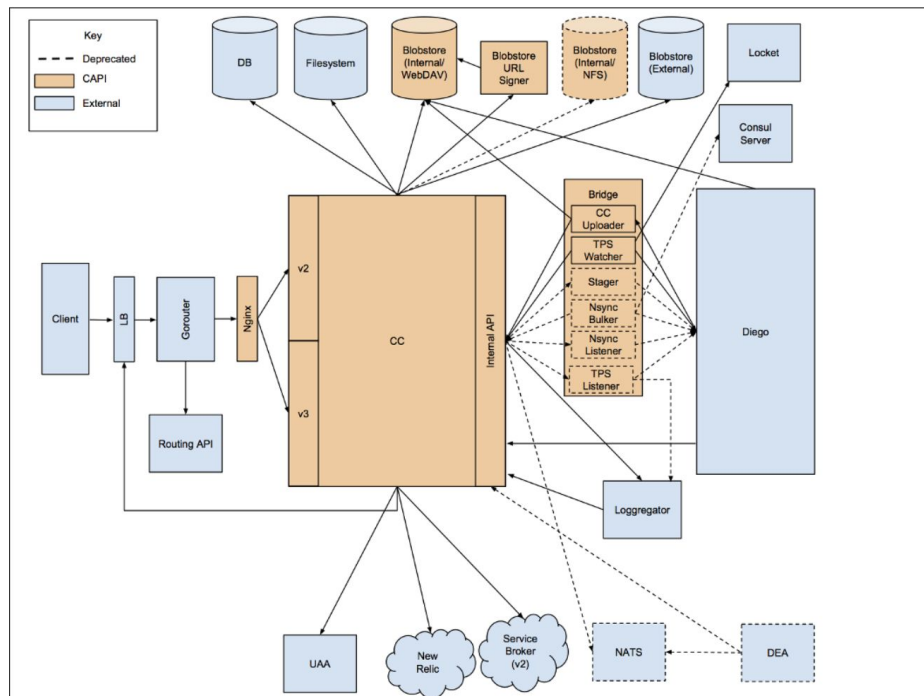


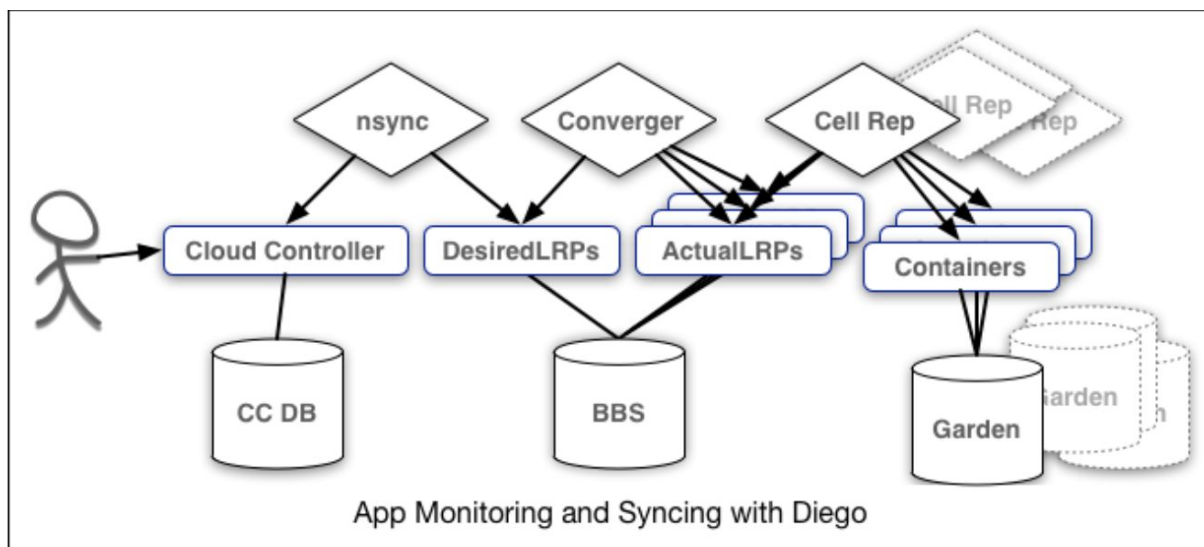
Figure 2.2: Architecture of Cloud Foundry

The Cloud Controller also maintain records of orgs, spaces, user roles, services, and more.

nsync, BBS, and Cell Reps

To keep applications available, cloud deployments must constantly monitor their states and reconcile them with their expected states, starting and stopping processes as required.

[8]



The nsync, BBS, and Cell Rep components work together along a chain to keep apps running. At one end is the user. At the other end are the instances of applications running on widely-distributed VMs, which may crash or become unavailable.

Here is how the components work together:

- **nsync** receives a message from the Cloud Controller when the user scales an app. It writes the number of instances into a **DesiredLRP** structure in the Diego BBS database.
- **BBS** uses its convergence process to monitor the **DesiredLRP** and **ActualLRP** values. It launches or kills application instances as appropriate to ensure the **ActualLRP** count matches the **DesiredLRP** count.
- **Cell Rep** monitors the containers and provides the **ActualLRP** value.

Services

Service Brokers

Applications typically depend on services such as databases or third-party SaaS providers. When a developer provisions and binds a service to an application, the service broker for that service is responsible for providing the service instance.

Messaging

Consul and BBS

Cloud Foundry component VMs communicate with each other internally through HTTP and HTTPS protocols, sharing temporary messages and data stored in two locations:

- A Consul server stores longer-lived control data, such as component IP addresses and distributed locks that prevent components from duplicating actions.
- Diego's Bulletin Board System (BBS) stores more frequently updated and disposable data such as cell and application status, unallocated work, and heartbeat messages. The BBS stores data in MySQL, using the Go MySQL Driver.

The route-emitter component uses the NATS protocol to broadcast the latest routing tables to the routers.

Metrics and Logging

Loggregator

The Loggregator (log aggregator) system streams application logs to developers.

Metrics Collector

The metrics collector gathers metrics and statistics from the components. Operators can use this information to monitor a Cloud Foundry deployment.

2.2 BOSH

BOSH is a project that unifies release engineering, deployment, and lifecycle management of small and large-scale cloud software. BOSH can provision and deploy software over hundreds of VMs. It also performs monitoring, failure recovery, and software updates with zero-to-minimal downtime.

While BOSH was developed to deploy Cloud Foundry PaaS, it can also be used to deploy almost any other software (Hadoop, for instance). BOSH is particularly well-suited for large distributed systems. In addition, BOSH supports multiple Infrastructure as a Service (IaaS) providers like VMware vSphere, Google Cloud Platform, Amazon Web Services EC2, and OpenStack. There is a Cloud Provider Interface (CPI) that enables users to extend BOSH to support additional IaaS providers such as Apache CloudStack and VirtualBox.

BOSH allows individual developers and teams to easily version, package and deploy software in a reproducible manner.

Any software, whether it is a simple static site or a complex multi-component service, will need to be updated and repackaged at some point. This updated software might need to be deployed to a cluster, or it might need to be packaged for end-users to deploy to their own servers. In a lot of cases, the developers who produced the software will be deploying it to their own production environment. Usually, a team will use a staging, development, or demo environment that is similarly configured to their production environment to verify that updates run as expected. These staging environments are often taxing to build and administer. Maintaining consistency between multiple environments is often painful to manage.

Developer/operator communities have come far in solving similar situations with tools like Chef, Puppet, and Docker. However, each organization solves these problems in a different way, which usually involves a variety of different, and not necessarily well-integrated, tools. While these tools exist to solve the individual parts of versioning, packaging, and deploying software reproducibly, BOSH was designed to do each of these as a whole.

BOSH was purposefully constructed to address the four principles of modern Release Engineering in the following ways:

Identifiability: Being able to identify all of the source, tools, environment, and other components that make up a particular release.

BOSH has a concept of a software release which packages up all related source code, binary assets, configuration etc. This allows users to easily track contents of a particular release. In addition to releases BOSH provides a way to capture all Operating System dependencies as one image.

Reproducibility: The ability to integrate source, third party components, data, and deployment externals of a software system in order to guarantee operational stability.

BOSH tool chain provides a centralized server that manages software releases, Operating System images, persistent data, and system configuration. It provides a clear and simple way of operating a deployed system.

Consistency: The mission to provide a stable framework for development, deployment, audit, and accountability for software components.

BOSH software releases workflows are used throughout the development of the software and when the system needs to be deployed. BOSH centralized server allows users to see and track changes made to the deployed system.

Agility: The ongoing research into what are the repercussions of modern software engineering practices on the productivity in the software cycle, i.e. continuous integration.

BOSH tool chain integrates well with current best practices of software engineering (including Continuous Delivery) by providing ways to easily create software releases in an automated way and to update complex deployed systems with simple commands.

2.3 STEMCELL

A stemcell is a versioned Operating System image wrapped with IaaS specific packaging.

A typical stemcell contains a bare minimum OS skeleton with a few common utilities pre-installed, a BOSH Agent, and a few configuration files to securely configure the OS by default. For example: with vSphere, the official stemcell for Ubuntu Trusty is an approximately 500MB VMDK file. With AWS, official stemcells are published as AMIs that can be used in your AWS account.

Stemcells do not contain any specific information about any software that will be installed once that stemcell becomes a specialized machine in the cluster; nor do they contain any sensitive information which would make them unable to be shared with other BOSH users. This clear separation between base Operating System and later-installed software is what makes stemcells a powerful concept.

In addition to being generic, stemcells for one OS (e.g. all Ubuntu Trusty stemcells) are exactly the same for all infrastructures. This property of stemcells allows BOSH users to quickly and reliably switch between different infrastructures without worrying about the differences between OS images.

The Cloud Foundry BOSH team is responsible for producing and maintaining an official set of stemcells. Cloud Foundry currently supports Ubuntu Trusty and CentOS 6.5 on vSphere, AWS, OpenStack, and vCloud infrastructures. Stemcells are distributed as tarballs. By introducing the concept of a stemcell, the following problems have been solved:

- Capturing a base Operating System image
- Versioning changes to the Operating System image
- Reusing base Operating System images across VMs of different types
- Reusing base Operating System images across different IaaS

3. User Account and Authentication

The UAA is a multi tenant identity management service, used in Cloud Foundry, but also available as a stand alone OAuth2 server.

OAuth is the industry-standard protocol for authorization that allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password. OAuth acts as an intermediary on behalf of the end user, providing the service with an access token that authorizes specific account information to be shared.

The UAA is an OAuth2 server that can be used for centralized identity management.

- owns the user accounts and authentication sources (SAML, LDAP)
- supports standard protocols such as SAML, LDAP and OpenID Connect to provide single sign-on and delegated authorization to web applications
- can be invoked via JSON APIs
- provides a basic login/approval UI for web client apps
- supports APIs for user account management for an external web UI
- most of the APIs are defined by the specs for the OAuth2, OpenID Connect, and SCIM standards.

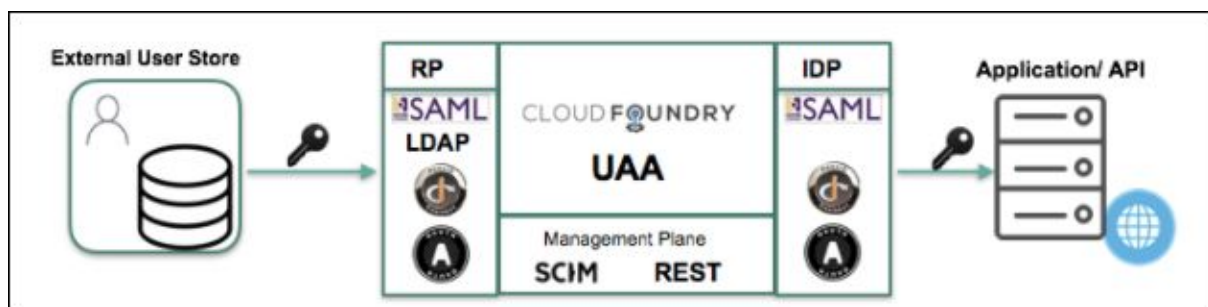


Figure 3.1: [4] Diagram representing the cloud foundry UAA architecture

In this section we cover the oauth flow for our system, starting from client application registration to final redirection to robocode application

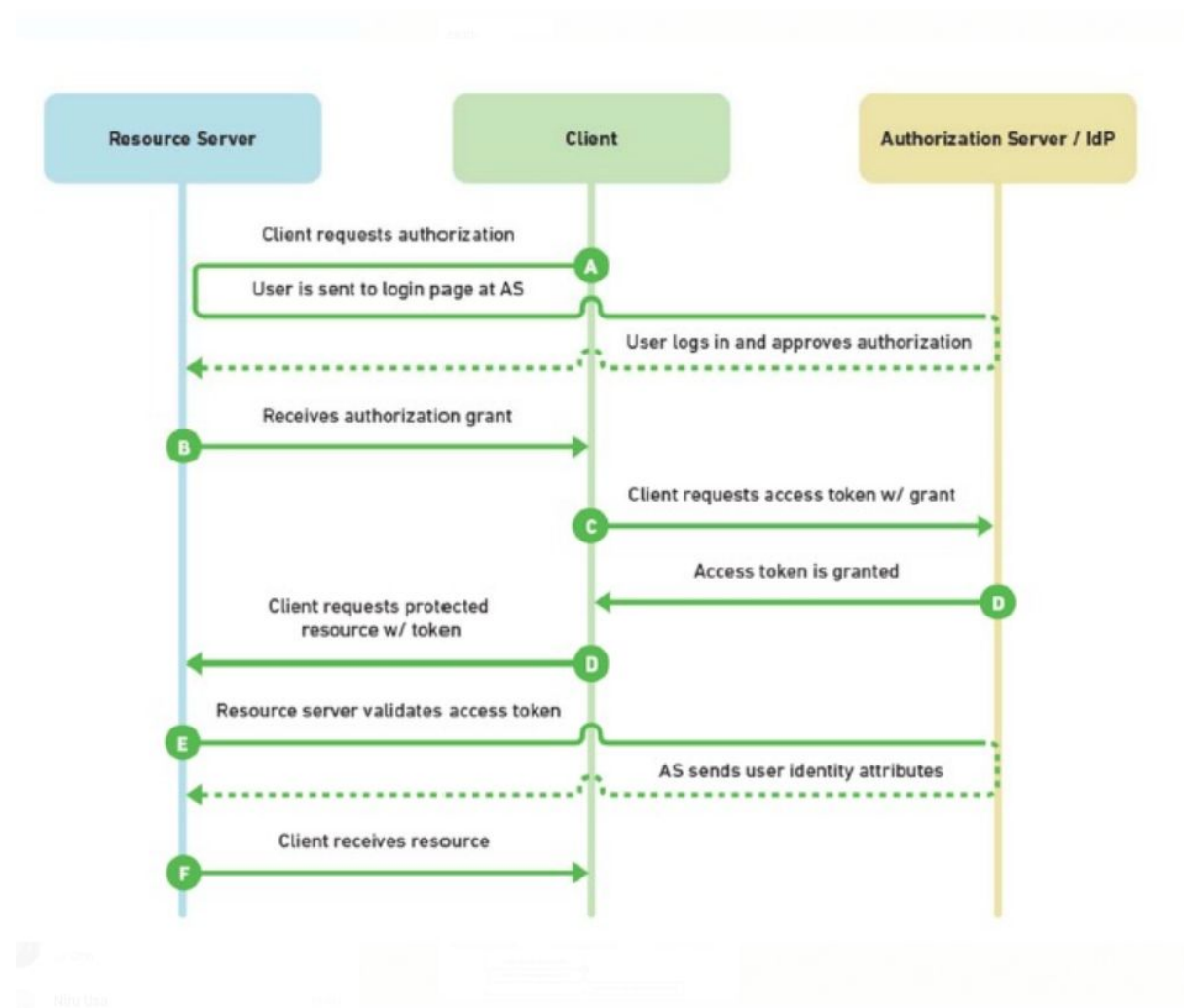


Figure 3.2: [1] Diagram representing the oauth flow.

- The client application is registered with the UAA server with specifications described in table 3.1
- All cloud foundry users are redirected from the client application home page to the UAA login page to verify their cloud foundry credentials
- If the user is authorized by the Authorization server, then the client application receives an authorization grant with the following parameters

Parameter	Description
Response type	a unique string representing the registration information provided by the client
Client id	a unique string representing the registration information provided by the client
scope	Permissions and level of access
Redirect uri	Redirection uri. In our system the redirection uri is set to redirect to the list of applications the user has access to

Table 3.1 : Authentication Grant Parameters

- Client Application then requests an access token from the authorization server (API) by presenting authentication of its own identity, and the authorization grant received previously.
- If the application identity is authenticated and the authorization grant is valid, the authorization server issues an access token to the client application.
- The client application requests the resource from the resource server (API) and presents the access token for authentication. The resource server API endpoints that we target are described.
- If the access token is valid, the resource server provides all requested resources within the scope of the request
- The user is finally redirected to our Robocode Application

We define next, what some of the scopes represent as well as all the **API Endpoints**, of the Resource Server, to target in order to fetch relevant information for our system.

3.1 Commands to register the client

- Uaac target `https://uaa.boshlite.com`
- Uaac token client get admin -s (uaa_Admin_client_secret)
- uaac client add --name triallogin --scope 'openid user_attributes roles scim.read cloud_controller.admin' --authorized_grant_types 'authorization_code,client_credentials' --access_token_validity 1209600 --authorities oauth.login --redirect_uri 'http://localhost:5000/profile'

3.2 Scopes

Scopes define permissions which are associated with users / a list of permissions that a client has on behalf of the end user. They are used by the *Authorization Server* to deny a token requested for a scope not on the list, and should be used by a Resource Server to deny access to a resource if the token has insufficient scope.

UAA covers a variety of scopes of privilege, including access to UAA, access to Cloud Controller, and access to the router. We define some of the scope accesses below [3]

Cloud Controller Scopes

cloud_controller.read	This scope gives the ability to read from any Cloud Controller route the token has access to.
cloud_controller.write	This scope gives the ability to post to Cloud Controller routes the token has access to.
cloud_controller.create	This admin scope gives full permissions to Cloud Controller.

Table 3.2 : Scopes of cloud controller

UAA Scopes

scim.write	This scope gives admin sufficient permissions to create accounts
scim.read	This scope gives admin read access to all SCIM endpoints, /Users, and /Groups.
scim.create	This scope gives the ability to create a user with a POST request to the /Users endpoint, but not to modify, read, or delete users.
openid	This scope is required to access the /userinfo endpoint. It is intended for OpenID clients.

Table 3.3: Scopes of UAA

3.3 Resource Server Information Retrieval

We retrieve the following information in order to implement role based access control policies for our system

1. Username of the current user logged in
2. Spaces to which the user belongs
3. Roles of this user in each of his/her spaces

Username Retrieval

The access token retrieved from the Authorization Server is a base 64 encoded JWT (JSON Web Token) string that contains three components, *header,payload,signature*.

We use a base64 decoder to decode the payload information in order to retrieve the username of the currently logged in user

Space/Org Information Retrieval

For accessing any resource information we perform a GET request to the respective API Endpoint using the acquired token from the Authorization Server. API Endpoint for obtaining information about spaces and orgs are provided below.

Role Information Retrieval

Cloud Foundry Resource Server provides us with information about all the users belonging to spaces and (or) organizations (to which the current user belongs to) for a particular role, for each role. To retrieve the roles of the current user we simply loop through all of the roles of orgs and spaces that the current user belongs to and find if the usernames under a role match the current user

API Endpoints

API	Description
https://api.bosh-lite.com/v2/spaces	Retrieve info of all spaces to which the current user belongs
https://api.bosh-lite.com/v2/organizations	Retrieve info of all organizations to which the current user belongs
https://api.bosh-lite.com/v2/apps	Retrieve info of all applications deployed in the spaces to which the user belongs

Table 3.4: End Points to target the resource server

4. Access Control Mechanism

CF uses a role-based access control (RBAC) system to grant Cloud Foundry users permissions appropriate to their role within an org or a space. Admins, Org Managers, and Space Managers can assign user roles using the cf CLI.

The Role-based access control mechanism of Cloud Foundry is used to achieve access control among multiple tenants using RoboCode application.

Org

An org is a development account that an individual or multiple collaborators can own and use. All collaborators access an org with user accounts. Collaborators in an org share a resource quota plan, applications, services availability, and custom domains.

By default, an org has the status of *active*. An admin can set the status of an org to *suspended* for various reasons such as failure to provide payment or misuse. When an org is suspended, users cannot perform certain activities within the org, such as push apps, modify spaces, or bind services. For details on what activities are allowed for suspended orgs

User Account

A user account represents an individual person within the context of a CF installation. A user can have different roles in different spaces within an org, governing what level and type of access they have within that space.

Spaces

Every application and service is scoped to a space. A space provides users with access to a shared location for application development, deployment, and maintenance. Each space role applies only to a particular space.

Roles and Permission

A user can have one or more roles. The combination of these roles defines the user's overall permissions in the org and within specific spaces in that org.

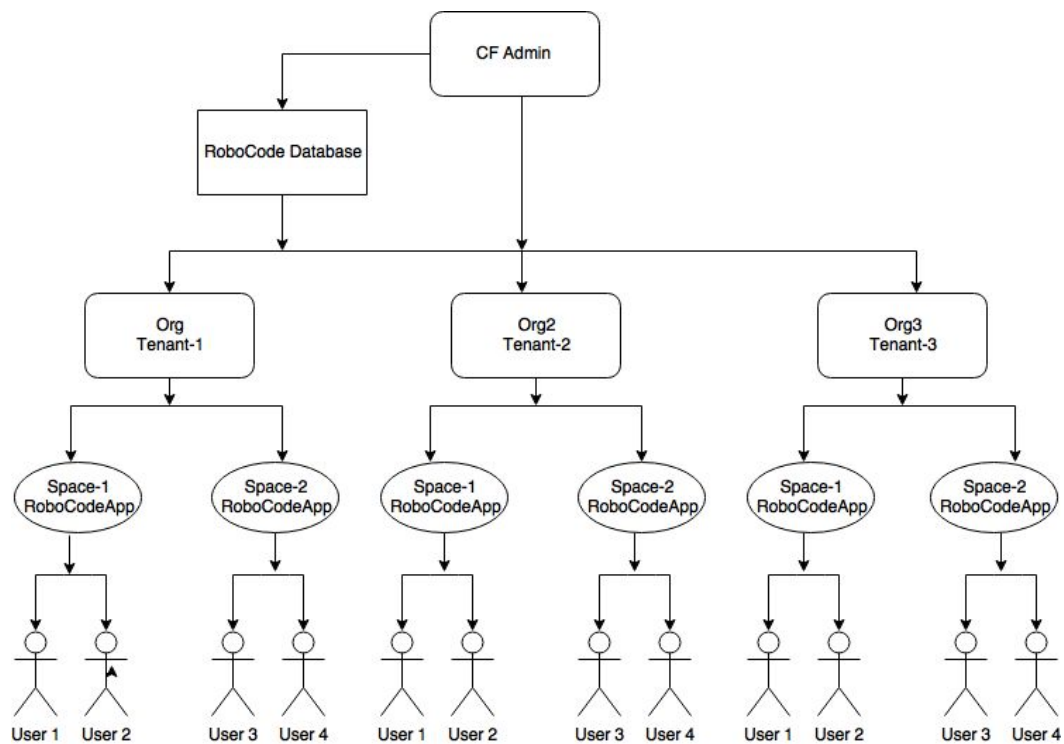
- **Admin** is a user role that has been assigned the `cloud_controller.admin` scope in UAA. An admin user has permissions on all orgs and spaces and can perform operational actions using the Cloud Controller API.
- **Admin Read-Only** is a user role that has been assigned the `cloud_controller.admin_read_only` scope in UAA. This role has read-only access to all Cloud Controller API resources.
- **Global Auditor** is a user role that has been assigned the `cloud_controller.global_auditor` scope in UAA. This role has read-only access to all Cloud Controller API resources except for secrets such as environment variables. The Global Auditor role cannot access those values.
- **Org Managers** are managers or other users who need to administer the org.
- **Org Auditors** view but cannot edit user information and org quota usage information.
- **Org Billing Managers** create and manage billing account and payment information.
- **Org Users** can view the list of other org users and their roles. When an Org Manager gives a person an Org or Space role, that person automatically receives Org User status in that Org.
- **Space Managers** are managers or other users who administer a space within an org.
- **Space Developers** are application developers or other users who manage applications and services in a space.
- **Space Auditors** view but cannot edit the space

4.1 Access Control Model for Robocode on Cloud Foundry

4.1.1 Access Control Model within each tenant

The Role based access control mechanism of Cloud Foundry is taken advantage to build and enforce the access control among Multiple Tenants of Robocode.

- The individual tenants of Robocode are assigned as ORGs in the CloudFoundry and the robocode application is pushed to spaces inside the ORG.
- An ORG could have multiple spaces for load balancing among users and every space would have an instance of the robocode application running.
- The ORGs share a common Database to maintain the user and application information.
- Privileges to the users are assigned through the ORG and SPACE roles provided by cloud Foundry.



Role	Permission
Admin CloudFoundry Admin	<ul style="list-style-type: none"> • Create new tenants • Delete existing tenants • Have global privileges and access • Scale up new spaces to tenants for load balancing
ORG Manager	<ul style="list-style-type: none"> • Create new user • Delete existing user • Set/Unset roles and access privileges for users(space manager, space auditor, space developer)
SPACE Developer	<ul style="list-style-type: none"> • User • Can read robots • Can edit robots • Can play Battle
SPACE Auditor	<ul style="list-style-type: none"> • User • Can read robots • Can play Battle






Role	Create Robot	Edit Robot	View Robot	Share Robots	Play Battle
Space Manager	✓	✓	✓	✓	✓
Space Developer	✓	✓	✓		✓
Space Auditor			✓		✓

- The Google's Link-share-within org mechanism has been used as a model for sharing robots with the users belonging to the same organisation.
- The Cloudfoundry user-roles and relationship table are used to enforce access control for robot sharing.
- All users are allowed to share their robots with users belonging to the same org(tenant).
- The relationship field in the relationship table is used to identify the mapping of robotID with the sharedwith userID.
- A user is allowed to play battle with his own robots and robots shared with him.
- A user is not allowed to edit the robots shared with him.

4.1.2 Cross Tenant Access Control

- The Robots of a user from one org(tenant) can be shared among users of other tenants.
- The tenants share a common database.
- If a user is willing to share his robots with specific users of other tenants he could set the **sharedWith** field in the database.
- The sharedWith field stores the information in the format [userID, orgID, spaceID]
 userID - ID of the user with whom the robot is to be shared
 orgID - orgID/TenantID of the user with whom the robot is to be shared
 spaceID - spaceID of the user with whom the robot is to be shared

Link sharing

- ☐  **On - Public on the web**
Anyone on the Internet can find and access. No sign-in required.
- ☐  **On - Anyone with the link**
Anyone who has the link can access. No sign-in required.
- ☐  **On - traderscolo.com**
Anyone at traderscolo.com can find and access.
- ☐  **On - Anyone at traderscolo.com with the link**
Anyone at traderscolo.com who has the link can access.
- ☒  **Off - Specific people**
Shared with specific people.

Note: Items with any link sharing option can still be published to the web. [Learn more](#)

Figure 4 : Google's Link sharing mechanism

4.2 Commands for Access Control

Login:

- `cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]`
- `cf auth USERNAME PASSWORD`
- `cf target [-o ORG] [-s SPACE]`

Listing users :

- `cf org-users`
- `cf space-users`

Managing Roles:

- `cf set-org-role USERNAME ORG ROLE`
- `cf unset-org-role USERNAME ORG ROLE`
- `cf set-space-role USERNAME ORG SPACE ROLE`
- `cf unset-space-role USERNAME ORG SPACE ROLE`

4.3 Implementation of Access Control

```

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404


@app.route('/login', methods=['GET'])
def login():
    """Step 1: User Authorization.
    Redirect the user/resource owner to the OAuth provider (i.e. Github)
    using an URL with a few key OAuth parameters.
    """
    cloudfoundry = OAuth2Session(client_id)
    authorization_url, state = cloudfoundry.authorization_url(
        authorization_base_url)

    # State is used to prevent CSRF, keep this for later.
    session['oauth_state'] = state
    session['logged_in'] = True
    session['uaa'] = baseUAAurl
    return redirect(authorization_url)


@app.route("/callback", methods=["GET"])
def callback():
    """ Step 3: Retrieving an access token.
    The user has been redirected back from the provider to your registered
    callback URL. With this redirection comes an authorization code included
    in the redirect URL. We will use that to obtain an access token.
    """

    cloudfoundry = OAuth2Session(client_id, state=session['oauth_state'])
    token = cloudfoundry.fetch_token(token_url, client_secret=client_secret,
                                     authorization_response=request.url,
                                     verify=False)
    session['oauth_token'] = token
    return redirect(url_for('.profile'))


@app.route("/profile", methods=["GET"])
def profile():
    """Fetching a protected resource using an OAuth 2 token.
    """

```

```

if not session.get('oauth_token'):
    return redirect(url_for('login'))
tokenString = "bearer {0}".format(session['oauth_token']['access_token'])
headers = {"Authorization": tokenString}
profileInfo = {'access_token': session['oauth_token']['access_token']}

```

```

# get user summary
userinfourl = '{}'/userinfo'.format(baseUAAurl)
userinfo = json.loads(requests.get(
    userinfourl, headers=headers, verify=False).text)
session['userinfo'] = userinfo
profileInfo['userinfo'] = json.dumps(session['userinfo'])

```

```

# Method 1 : get user roles by orgs and space
usersummaryurl = '{0}/v2/users/{1}/summary'.format(
    baseAPIurl, userinfo['user_id'])
usersummary = json.loads(requests.get(
    usersummaryurl, headers=headers, verify=False).text)

```

```

if usersummary.get('entity'):
    spaceWiseUserRoles = getSpaceWiseUserRoles(usersummary['entity'])
else:
    # Method 2 : get user roles by orgs and space
    spaceWiseUserRoles = {}
    spaceurl = baseAPIurl + '/v2/spaces'
    spaceresponse = requests.get(spaceurl, headers=headers, verify=False)
    space_json_data = json.loads(spaceresponse.text)
    for spaceresource in space_json_data['resources']:
        entity = spaceresource['entity']
        spaceGuid = spaceresource['metadata']['guid']

```

```

# get all auditors
auditorurl = baseAPIurl + entity['auditors_url']
auditorresponse = json.loads(requests.get(
    auditorurl, headers=headers, verify=False).text)
if isInThisRole(auditorresponse, userinfo['user_name']):
    spaceWiseUserRoles[spaceGuid] = {
        'role': 'auditor',
        'name': spaceresource['entity']['name']
    }

```

```

# get all developers
devurl = baseAPIurl + entity['developers_url']
devresponse = json.loads(requests.get(
    devurl, headers=headers, verify=False).text)

```

```

        if isInThisRole(devresponse, userinfo['user_name']):
            spaceWiseUserRoles[spaceGuid] = {
                'role': 'developer',
                'name': spaceresource['entity']['name']
            }

    # get all managers
    managerurl = baseAPIurl + entity['managers_url']
    managerresponse = json.loads(requests.get(
        managerurl, headers=headers, verify=False).text)
    if isInThisRole(managerresponse, userinfo['user_name']):
        spaceWiseUserRoles[spaceGuid] = {
            'role': 'manager',
            'name': spaceresource['entity']['name']
        }

profileInfo['spaceWiseUserRoles'] = json.dumps(spaceWiseUserRoles)
session['spaceWiseUserRoles'] = spaceWiseUserRoles

# get user apps from all spaces
url = '{} /v2/apps'.format(baseAPIurl)
response = requests.get(url, headers=headers, verify=False)
appsData = json.loads(response.text)
appsUrls = {}

# user accessible app url
for resource in appsData['resources']:
    routes_url = baseAPIurl + \
        resource['entity']['routes_url']
    routes_url_response = json.loads(requests.get(
        routes_url, headers=headers, verify=False).text)
    for app in routes_url_response['resources']:
        hostname = app['entity']['host']
        appsUrls[hostname] = {
            'url': 'http://{}/local.pcfdev.io'.format(hostname),
            'space_guid': app['entity']['space_guid'],
            'userRole': getSpaceRole(spaceWiseUserRoles, app['entity'][
                'space_guid'], userinfo['user_name'])}
profileInfo['apps'] = appsUrls

organization_guid = getOrganizationId(session, appsData)
profileInfo['org_id'] = organization_guid
profileInfo['org_users'] = json.dumps(getOrganizationUsers(
    session, organization_guid))
return render_template('profile.html', data=profileInfo)

```



```
def getSpaceRole(spaceWiseUserRoles, spaceguid, username):  
    for id in spaceWiseUserRoles:  
        if id == spaceguid:  
            return spaceWiseUserRoles[id]['role']
```

```
def isInThisRole(rolerresponse, username):  
    for resource in rolerresponse['resources']:  
        if resource['entity']['username'] == username:  
            return True
```

```
    return False
```

```
def getSpaceWiseUserRoles(entity):  
    spaceWiseUserRoles = {}  
  
    for subentity in entity['spaces']:  
        spaceWiseUserRoles[subentity['metadata']['guid']] = {  
            'role': 'developer',  
            'name': subentity['entity']['name']  
        }  
    for subentity in entity['managed_spaces']:  
        spaceWiseUserRoles[subentity['metadata']['guid']] = {  
            'role': 'manager',  
            'name': subentity['entity']['name']  
        }  
    for subentity in entity['audited_spaces']:  
        spaceWiseUserRoles[subentity['metadata']['guid']] = {  
            'role': 'auditor',  
            'name': subentity['entity']['name']  
        }
```

```
    return spaceWiseUserRoles
```

```
def getOrganizationUsers(session, organization_guid):  
    tokenString = "bearer {0}".format(session['oauth_token']['access_token'])  
    headers = {"Authorization": tokenString}  
    orgusersurl = baseAPIurl + '/v2/organizations/{}/users'.format(  
        organization_guid)  
    orgusersresponse = json.loads(requests.get(  
        orgusersurl, headers=headers, verify=False).text)  
    orgusers = {}  
    if orgusersresponse['resources']:
```

```

    for orguser in orgusersresponse['resources']:
        orgusers[orguser['entity']]['username'] = orguser['metadata']['guid']
    return orgusers

```

```

def getOrganizationId(session, app_Data):
    tokenString = "bearer {0}".format(session['oauth_token']['access_token'])
    headers = {"Authorization": tokenString}
    spaceguid = app_Data['resources'][0]['entity']['space_guid']
    spaceUrl = baseAPIurl + '/v2/spaces/{0}'.format(spaceguid)
    spaceinfo = json.loads(requests.get(
        spaceUrl, headers=headers, verify=False).text)
    orgId = spaceinfo['entity']['organization_guid']
    return orgId

```

In Robocode,

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    if (session.getAttribute("cfaccess_token") == null)
    {
        response.sendRedirect("https://localhost:5000");
        Return;
    }
    RequestDispatcher requestDispatcher = request.getRequestDispatcher("welcome.jsp");
    requestDispatcher.forward(request, response);
}

```

5. Database Schema

We have deployed two different database services in our cloud foundry system.

1. MySQL Deployed using Docker container
 2. p-mysql which acts as a standalone database service
- To create a database, we first generate a service key for a database service instance. The key contains the following fields:
{dbname,host,password,port,username}
 - To connect this service instance from our application we connect to the mysql database with the above credentials
 - The original robocode did not have the feature of creating packages to new users.
 - A new “**userpackages**” table has been added and the code was modified to enable new users to create packages if they do not have one.
 - The “**relationship**” table maintains the information about the shared robots. “**robotID**” of the shared robots are mapped to the sharedwith user’s “**userid**”.

5.1 Schema

Robocode Robot	
id	: bigint(20)
userID	: varchar(80)
packageID	: varchar(80)
dataaccess	: varchar(80)
filepath	: longtext
CreateDate	: varchar(200)
UpdatedDate	: varchar(200)
RoboCode	: longtext

v		Robocode relationship
		relationships : bigint(200)
		robotID : varchar(200)
		userid : varchar(200)

v		Robocode userpackages
		userid : varchar(50)
		packagename : varchar(40)

6. Performance Analysis

Proposed performance analysis

The Cloud Foundry Routing subsystem handles and forwards all incoming requests to platform system components and applications. As such, the latency and throughput of the routing components has significant impact on the performance of the applications. Seemingly harmless code changes to these components could lead to unexpected performance regressions. While the main goal is to find and fix regressions, performance testing also enables to establish a baseline benchmark for the routing tier, and how to scale the deployment.

The load generating tool used for performance setup is a simple utility called hey. Hey is very similar to ApacheBench (ab) in its ease of use. It does not require as complicated of a setup as other load generating frameworks. It is also easy to parse the output for automated testing so tests could easily be run in a continuous integration pipeline.

“hey” runs provided number of requests in the provided concurrency level and prints stats

We also use “boom” to send multiple requests in the provided concurrency level and print stats. We send 100 requests to the application endpoints to test their performance.

```
----- Results -----
Successful calls           0
Total time                 4.4944 s
Average                   0.0000 s
Fastest                   0.0000 s
Slowest                   0.0000 s
Amplitude                 0.0000 s
Standard deviation        0.000000
RPS                       0
BSI                       :(

----- Status codes -----

----- Legend -----
RPS: Request Per Second
BSI: Boom Speed Index
groupc7@groupc7:~$
```

Fig : Performance Analysis for Robocode application running locally on Tomcat Server

```

groupc8@groupc8-OptiPlex-9020:~$ boom http://robocodev3.bosh-lite.com/app/welcome.jsp -c 10 -n 100
Server Software: Unknown
Running GET http://10.244.0.34:80/app/welcome.jsp
Host: robocodev3.bosh-lite.com
Running 100 queries - concurrency 10
[=====>] 100% Done

----- Results -----
Successful calls      100
Total time           0.3807 s
Average              0.0352 s
Fastest              0.0050 s
Slowest              0.1409 s
Amplitude             0.1359 s
Standard deviation    0.018333
RPS                  262
BSI                  Pretty good

----- Status codes -----
Code 200              100 times.

----- Legend -----
RPS: Request Per Second
BSI: Boom Speed Index
groupc8@groupc8-OptiPlex-9020:~$ boom http://robocodev3.bosh-lite.com/app/welcome.jsp -c 10 -n 100

```

Fig : Performance analysis for Robocode application running on CloudFoundry

The runtime have been better for robocode application running on cloudfoudry, since cf could do load balancing and bring up instances of the application on the fly.

Since cloudFoundry uses load-based scaling its efficiency is better. It would scale-up on high load and scale-down when there is low load.

Security disadvantages in Original Robocode :

- Users had access to all of the robots in the database.
- There is no access control policy to restrict robots to particular user.
- The application was less secure - users were authenticated using the password login mechanism in the robocode

Security Enhancement in CF Robocode :

- The robots are restricted to specific users using the role based access control mechanism of cloudfoundry.
- Security is enhanced - Users are authenticated using the cloudfoundry's authentication server and the api endpoints are hidden.

References

- [1] <https://docs.apigee.com/sites/docs/files/oauth-auth-code-flow%20%281%29.png>
- [2] <https://github.com/cloudfoundry/uaa/blob/master/docs/UAA-Security.md>
- [3] <https://docs.cloudfoundry.org/concepts/architecture/uaa.html#uaa-scopes>
- [4] <https://docs.cloudfoundry.org/uaa/uaa-overview.html>
- [5] <https://www.openstack.org/software/>
- [6] <https://maas.io/how-it-works>
- [7] <http://docs.cloudfoundry.org/deploying/index.html>
- [8] <https://docs.cloudfoundry.org/concepts/architecture/>
- [9] <https://docs.cloudfoundry.org/concepts/architecture/uaa.html>
- [10] <https://docs.cloudfoundry.org/concepts/architecture/cloud-controller.html>
- [11] <https://docs.cloudfoundry.org/concepts/roles.html#roles>