**SPRING CLOUD CLI** – Password Encryption and Decryption in config server:

**STEPS**:

Download spring Boot Cli -spring-boot-cli-2.0.4.RELEASE and set env path in pc

**Step1**:

After installation check in Command Prompt

1.1.spring –version

**Step2**:

2.1.spring install org.springframework.cloud:spring-cloud-cli:1.3.2.RELEASE

Check version installed:

2.2.spring cloud --version

**Step3**:

Execute this command for encrypt any credentials with any encryption key

3.1. spring encrypt anypassword --key mysecretkey

U will get encrypted like:e.g 2c9c7e7da0dc894bf2618a9d4877398dee9046806359d487633c18e9225d2e97

To Decrypt and check:

spring decrypt --key mysecretkey 2c9c7e7da0dc894bf2618a9d4877398dee9046806359d487633c18e9225d2e97

**Step4**:

Integration with application:

4.1: Add dependency in pom

<!-- Spring Cloud CLI password decryption dependency for config server -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-rsa</artifactId>

</dependency>

4.2: Add the encrypted value in config server like this:

{cipher}2c9c7e7da0dc894bf2618a9d4877398dee9046806359d487633c18e9225d2e97

Config server decrypts if any value is appended with {cipher}.

4.3: Pass the encryption key to application which we used to encrypt from ENV

**ENCRYPT\_KEY =mysecretkey**

**Connection Manager:**

**Solution 1:**

**Get the sqlhost file as URI from Static file server app from PCF(Static build pack).**

**Solution 2:**

**Have sqlhost file in classpath of the application and load it to the database driver.**

**SPRING CLOUD CONFIG SERVER:**

**Step 1:**

Create a config server for each space in PCF by providing the GIT repo details to access the properties .



Bind the service to the application.

Step 2: Add the necessary spring cloud dependencies. Ref: <https://westcorp-apptx-cookbook.app.pcfdev.one.west.com/recipes/config-server/>

Step 3: User Actuator to refresh the all the instances.

cf create-service p-config-server standard icrm-gateway-config-server -c {"git": { "uri": "git@repo.west.com:Thanga01/iCRM-gateway-repo.git", "label": "master","privateKey":"-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEAtZpMe7oUz9Lc5ShCS7T3As+DQAPTSKABwVeMrbpmlr9HNnmR\nZUugtnnA6ncsiAmI+i5h3UMUC/Y/sXvI3/+ghKjbpFCO1StjrcSVgzZ7CS7ydCBU\nK8QO4tybXdzFWNyuU2+Bv/oK9XABGrAeKpAKBEu4a1YOyeJRpW/WCTuafXbtmsjB\nnR2dGggdpWxwiWXDdPQzQXRqS88XbCP30paT9mjwrncX5kc7lYwQECmc7n/59YR6\np1gSHdiT37CboALWiLhL/hmunEMj9cNzGchshym8Qmf057iEQzh9kgazjKY1Fvje\nWXV2LWz4ubKQI+HHTq2tnLUIyoEGPc4zQqRk5wIDAQABAoIBAHjN58r2ghmtn5wh\nJ9jMzPRlrl3LLaXGKuS+pKp8sPnezDSEsh+8K3C3lYljNZBZJ3GloRxbHt7GNKCQ\nQ5QerF0TJukrXZiYgHp7pynd+AKn5EkzqEpk5bIsJn5N+kmeo6iFKL7YSqnylAl4\n1thhcQhiho+Ac94AW2e9+ZeyupZyg4F9NtfD2RYe2xaOl6o4V5KOEYnON8y9ar3p\nys97Bld+JIZFwE3VTDh+ndEqRVeNChQa06ZZF2SeGkErQFGmDVBYW4GAbOWR1Oto\njDpnfKM1cRRvb74lIR9w3sa2gt1pXOHnSj9qDzJ6CpHA9pC/AM3jS4IjVrL/Rcm6\ntJpq4QECgYEA4Tnjv82akdgpRYMxi1Dk/bbjEv52MYP0wdVKgiH8Rm9mPZP4D3n5\ncW4BqfUS3FRO9daPQxL8+nM3FjmoEA1PPlq6IeXCt6RanMppufjJ8jzh8NPNkyi5\nVO93q6igoLm0a8px0TmWkCKZHT1fFCZAWFH6aFQmVAzq4ySHvi/nfSECgYEAzmqF\nRDTfOx12vJLS1CnSnnPtimILz+k/CDjwkWwTigrNE1VANbYMSr4tszgww0TwJ7L5\n9pSKFR6nmrtCDz5jRS5b3Epk1RAb2+h+cAoGKf2yMgSw2/JoZMiMnaEF/yon0dRc\nIxgAEoaytYeUbYkMagCZSMZYNUFeRu+moOtf2QcCgYBjrQcKP6gEm7HIPcOTcH2n\ne2ULZupQB6IuloKfDQNk+LA62F2UG9s1RAYgfYGXWQiI7CgC+0hSXuMmRuYGFJ3Z\nbrWpgtGmDGq9sScRUvWs6xzjRBla408CM+i2Y66S8xzSB4pPmjj0GRvCQQLvG+6m\nJ8G2omdEE2Nwmx4oIXNFgQKBgQCX1vG7T61aXf0nEFY4LZ+6YYq+Pgt0YJ62XLRS\nNQ9Wf0WF0DNQj18RaD5T1KTCL6b9fQG1KmU8zfT6tH8OmZUKoeR+8tajiJsFtEu1\n/Rm1r+JyOtMiqHm0kXYmyfmQTyFFkBJK2m/LMPyq3lYIEUVpgJFBYG4olrJUOhv3\nyc0S5wKBgAG3ctL9DxcQaB6zbl7MnxQpymW+kPEaMW6/WpwaulJKUfpaZ8Fi0W7X\nGaojobhItJP9GvpuAClad0SupWrDxr0a9gMFQvGyj64Gc3Kv2OecpsJ5/fqasvHT\ngaG5+brA2w19QtJbipFI7Hp53Tc2n9rRDpxyddmGYt6ptZ+PhJCN\n-----END RSA PRIVATE KEY-----\n" } }

{"git": { "uri": "git@repo.west.com:karthi01/aceapi-config.git", "privateKey": "" } }

User Provided Service(CUPS):

Refer the mail Attachment.



Static Build Pack to serve static files:

Step1:

Place the static files in a folder with the structure as it is in icrm App.

Step2:

Push to PCF from the folder using command:

**cf push icrm-r-serve-static-content-qa -b staticfile\_buildpack**

Refer: <https://docs.cloudfoundry.org/buildpacks/staticfile/index.html>

Spring cloud gateway:

Spring Cloud gateway was suggested to be the edge proxy for IOL. Reference: https://cloud.spring.io/spring-cloud-gateway/

Built on Spring Framework 5, Project Reactor and Spring Boot 2.0 Able to match routes on any request attribute. Predicates and filters are specific to routes. Hystrix Circuit Breaker integration. Spring Cloud DiscoveryClient integration Easy to write Predicates and Filters Request Rate Limiting Path Rewriting

The following is the IOL Gateway artefacts created:

<https://repo.west.com/Thanga01/Mexico-Dojo-Thanga/tree/master/icrm-gateway>

<https://repo.west.com/Thanga01/iCRM-gateway-repo/blob/master/icrm-gateway.yml>

Blue-green with Spring cloud gateway:

Doing blue-green with Eureka requires that spring.application.name is changed during deployments to ICRM-Blue or ICRM-Green.

You can refer to some of the best practices here:

https://github.com/ryanjbaxter/bluegreen#running-the-sample-on-cloud-foundry (related links here: https://github.com/spring-cloud/spring-cloud-n )etflix/issues/1851

For polyglot service discovery services from PCF 2.2 onwards the below can be used: (This appears to be a much cleaner option though)

https://www.cloudfoundry.org/blog/polyglot-service-discovery-container-networking-cloud-foundry/

**Service Registry with Spring cloud discovery client:**

https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka

**Check session replication using PCC/Redis:**

**Enable Session State Caching with the Java Buildpack**:

When the service instance name is followed by the session-replication tag, the Java buildpack will download all the required resources for session state caching.

To enable session state caching, do one of the following:

* When creating your service instance name, append it with the session-replication tag. For example, for the p-cloudcache service:

$ cf create-service p-cloudcache my-service-instance -t session-replication

* When updating your service instance name (for example, if the updated name is new-service-instance), append it with the session-replication tag:

$ cf update-service new-service-instance -t session-replication

* End the service instance name with the text -session-replication: my-service-instance-session-replication.

**NFS mount:**

<https://westcorp-apptx-cookbook.app.pcfdev.one.west.com/recipes/nfs-volume-service-setup/>

**BPMA Road MAP:**

1.Remove JBoss dependencies and try to run in Tomcat 8.5 in local.

2.Identify the url's and get the firewall issues resolved for PCF.

3.Session Scope - Should use Distributedcache or RedisCache or PCC (Refer POC from ICRM)

4.Avoid using local file system to write files.

5.Change the xml configuration to java properties.

6.Logging - Logs should be written in stream. (Refer ICRM)

7.Property config - Use external configuration server.(Refer POC from ICRM Gateway)