Cloud Native Applications in Azure supporting Hybrid Cloud – Part II

Installing SUSE Cloud Application Platform on Azure AKS:

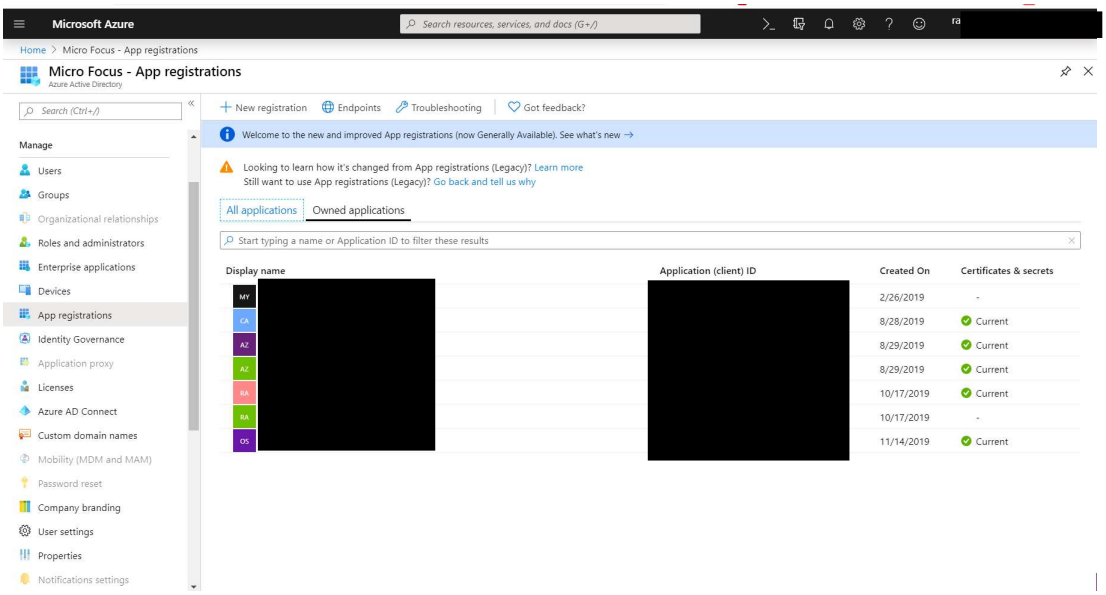
1. From your machine install az cli:
    - `sudo zypper addrepo https://download.opensuse.org/repositories/devel:languages:python:backports/SLE_15_SP1/devel:languages:python:backports.repo`
    - `sudo zypper refresh`
    - `sudo zypper install python-virtualenv`
    - `sudo zypper install -y curl`
    - `sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc`
    - `sudo zypper addrepo --name 'Azure CLI' --check https://packages.microsoft.com/yumrepos/azure-cli azure-cli`
    - sudo zypper install --from azure-cli -y azure-cli
2. Install Kubectl:

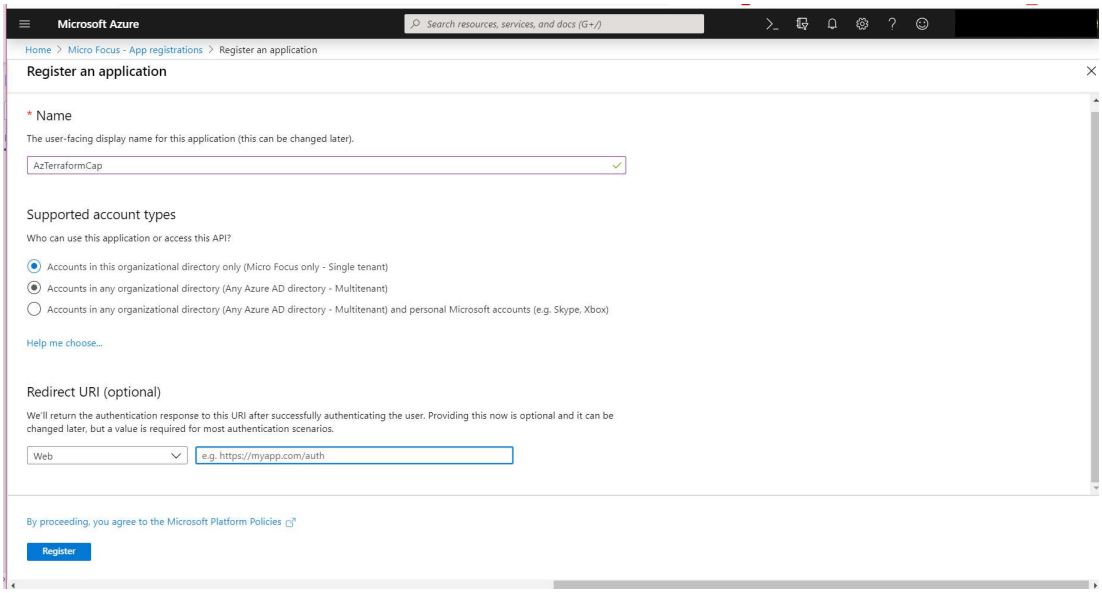
    ```
    curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
    chmod +x ./kubectl
    sudo mv ./kubectl /usr/local/bin/kubectl
    ```

3. Install jq and sed:
    - sudo zypper addrepo https://download.opensuse.org/repositories/utilities/SLE_15_SP1_Backports/utilities.repo
    - sudo zypper refresh
    - sudo zypper install jq
    - sudo zypper install sed
4. Create service principle in Azure, which will be used in the Terraform script to provision the AKS cluster and deploy SUSE Cloud Application Platform on it:

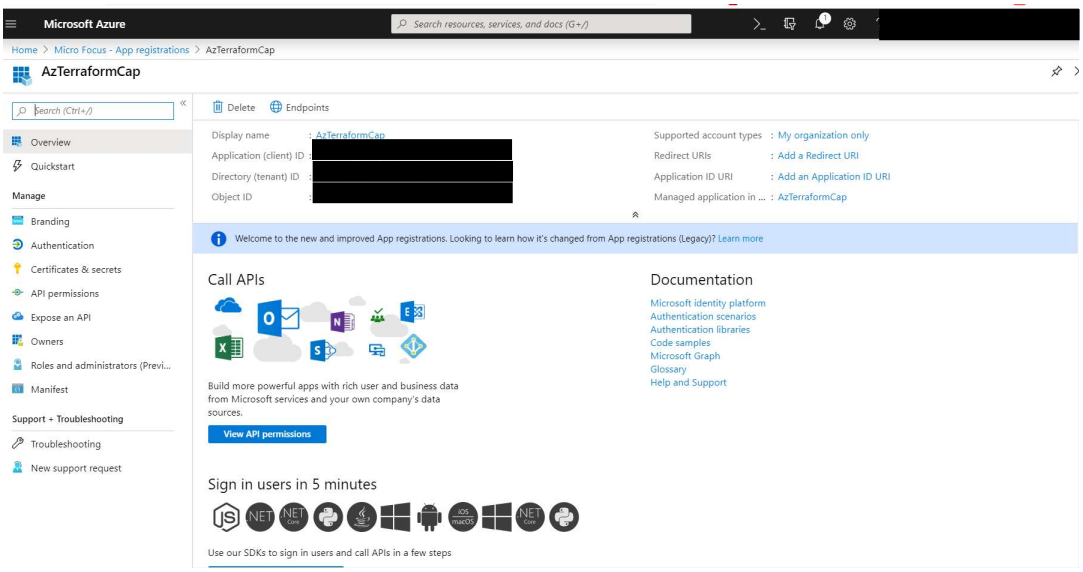
    We can do it from Azure portal or from the az cli. In this example, let us do it from the portal:

- Login to the portal and navigate to the Azure Active Directory then navigate to the app registrations side link. Click on New Registration and create a new AzTerraformCap account:

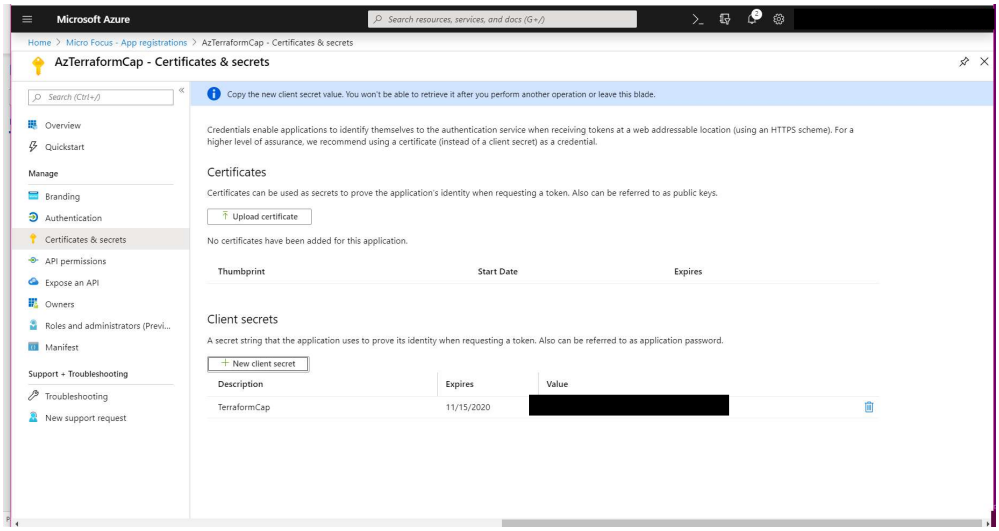




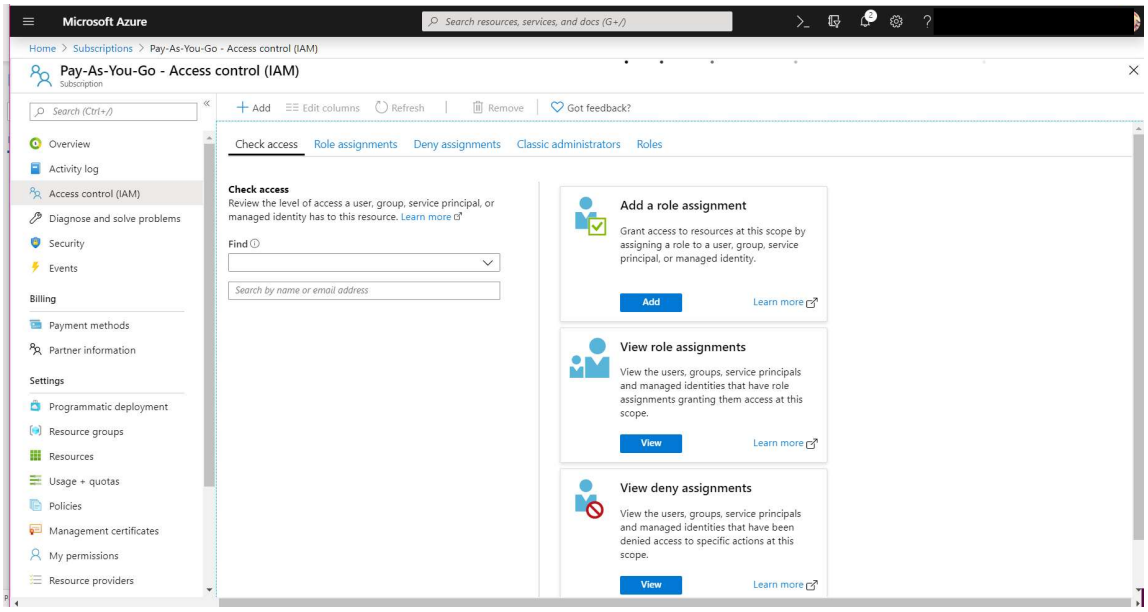
- Now Capture the Client ID and Tenant ID as they will be used in our configurations

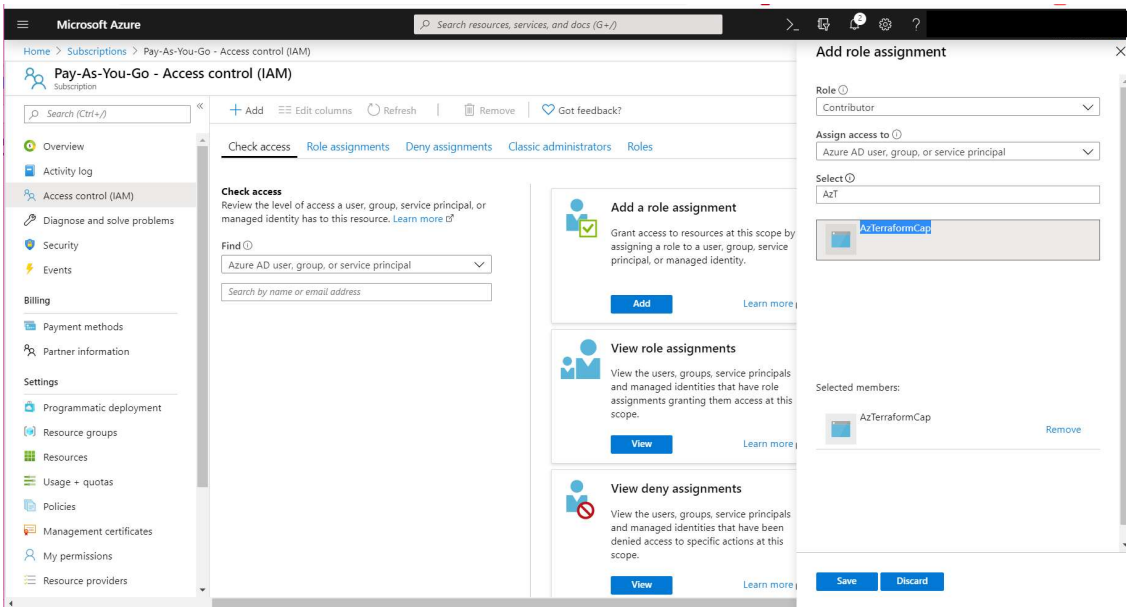


- Then navigate to Certificates & secrets to create secrets. Click new client secret and add a description called TerraformCap. Copy the secret value as we will be using it in configuring the Terraform scripts:



- Navigate to subscription then click on IAM Add a role assignment, selecting the contributor as a role and the user as AzTerraformCap:





5. Install cf cli, the cloud foundry cli that you can use in operation and development activities on SUSE Cloud Application Platform:

    - sudo SUSEConnect --product sle-module-cap-tools/15.1/x86_64
    - sudo zypper install cf-cli

6. Install Terraform

    a) Navigate to https://www.terraform.io/downloads.html and find the latest version of Terraform. At the time of writing this blog, the latest was 0.12.15.

    b) Download the zip:

    wget https://releases.hashicorp.com/terraform/0.12.15/terraform_0.12.15_linux_amd64.zip

    c) Unzip the zip file:

sudo unzip terraform_0.12.15_linux_amd64.zip

d) sudo mv terraform /usr/local/bin/

7. Install git:

   a) sudo zypper in git-core

8. Install Helm:

   - curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh
   - chmod +x get_helm.sh
   - ./get_helm.sh
   - helm init --client-only

9. Clone the Terraform scripts:

   git clone https://github.com/Rasadus03/cap-terraform.git -b e2e-cap-deploy

10. Navigate to cap-terraform/aks.

11. Run az login to login to azure using your account. Copy and paste the url and wait until you see the subscription details as the one shown below:

```
[
 {
  "cloudName": "AzureCloud",
  "id": "xxxxxxxxxxxxxxxxxxxxxxx",
  "isDefault": true,
  "name": "Pay-As-You-Go",
  "state": "Enabled",
  "tenantId": "xxxxxxxxxxxxxxxxxxxxx",
  "user": {
   "name": "xxxxxxxxxxxxxxx",
   "type": "user"
  }
 }
]
```

12. Set the following variable:

    export ARM_CLIENT_ID=*this is the client/app id for the service principle account we created earlier*

export ARM_CLIENT_SECRET= *this is the secret value we generated for the service principle account we created earlier*

export ARM_TENANT_ID= *this is the tenant id (it is the same as the one listed here) for the service principle account we created earlier*

export ARM_SUBSCRIPTION_ID= *this is the id you got in the JSON displayed after successful az login*

13. Edit "terraform.tfvars.json" to configure the Terraform variables:

{

"location": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *this is the region AKS cluster will be provisioned at, we will eastus*

"az_resource_group": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *this is the resource group in Azure tenant which will hold all the created nodes and the K8s cluster, we are going to use azterraformcap, it must be all in small letters and make use it is created in the same region selected for the location*

"ssh_public_key": "/xxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxx.pub", ← *this is the pub key generated you can use openssl to generate the key using ssh-keygen, this key is used to ssh to the worker nodes. It must be the full absolute path of the file, relative path is not accepted.*

"agent_admin": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *user used in logging into the worker nodes*

"client_id": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *same client id used in the environment variables*

"client_secret": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *same secret id used in the environment variables*

"cluster_labels": {},

"azure_dns_json": "/xxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxx/azure_dns_json.json", ← *path of the json file hold the DNS Zone configurations. It must be the full absolute path of the file, relative path is not accepted.*

"chart_values_file": "/xxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxx/scf-config-values.yaml", ← *path of the SUSE Cloud Application Platform configurations. It must be the full absolute path of the file, relative path is not accepted.*

"stratos_metrics_config_file": "/xxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxx/stratos-metrics-values.yaml", ← *path of the SUSE Cloud Application Platform metrics configurations. It must be the full absolute path of the file, relative path is not accepted.*

"disk_size_gb": "100", ← this is the worker node storage capacity.

"k8s_version": "1.14.8", ← this is the K8s version, please always note that the used version must be higher than 1.10 and must be supported by the selected region.

"node_count": "2", ← this is the number of worker nodes.

"machine_type": "Standard_D4s_v3", ← this is the node type used for the provisioned k8s workers/minion.

"cap_domain": "xxxxxxx" ← this is the domain variable that will be used in the scf-config-values.yaml, in our case it is demo.suseazurecap.org.

}

Here is an example to the terraform.tfvars.json file:

{

"location": "eastus",

"az_resource_group": "azterraformcap",

"ssh_public_key": "/home/rmohamed/cap-terraform/aks/azterraformcap.pub",

"agent_admin": "capazadmin",

"client_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",

"client_secret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",

"cluster_labels": {},

"azure_dns_json": "/home/rmohamed/cap-terraform/aks/azure_dns_json.json",

"chart_values_file": "/home/rmohamed/cap-terraform/aks/scf-config-values.yaml",

"stratos_metrics_config_file": "/home/rmohamed/cap-terraform/aks/stratos-metrics-values.yaml",

"disk_size_gb": "100",

"k8s_version": "1.14.8",

"node_count": "2",

"machine_type": "Standard_D4s_v3",

"cap_domain": "demo.suseazurecap.org"

}

14. Edit "azure_dns_json.json" to configure the used DNS Domain, the domain must be registered using the registrar service, for simplicity we will use Azure App Service Domains service, we will create a domain with the name suseazurecap.org and its associated resource group as suseazurecapnoterraform:

{

"tenantId": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *same tenant id used in the environment variables.*

"subscriptionId": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *same subscription id used in the environment variables.*

"resourceGroup": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *resource group we used for the newly created domain, in our case it is suseazurecapnoterraform.*

"aadClientId": "xxxxxxxxxxxxxxxxxxxxxxxxx", ← *the client id used in the environment variable*

"aadClientSecret": "xxxxxxxxxxxxxxxxxxxxxxxxx" ← *the secret value used in the environment variable*

}

Here is an example of the azure_dns_json.json file:

{

"tenantId": "xxxxxxxxxxxxxxxxxxxxxxx",

"subscriptionId": "xxxxxxxxxxxxxxxxx",

"resourceGroup": "susecapdemo",

"aadClientId": "xxxxxxxxxxxxxxxxxxxxxxxxxx",

"aadClientSecret": "xxxxxxxxxxxxxxxxxxxxxxxxx"

}

15. cp le-prod-cert-issuer.yaml.template le-prod-cert-issuer.yaml
16. Now edit the le-prod-cert-issuer.yaml to configure the certification authority parameters. The Terraform scripts are using by default the acme prod for generating secured certifications:

apiVersion: certmanager.k8s.io/v1alpha1

kind: ClusterIssuer

metadata:

name: letsencrypt-prod

spec:

acme:

Replace with your email address so you can be notified of expiring certificates

email: rania.mohamed@suse.com ← replace it with your mail

This is using letsencrypt production so beware of rate-limiting!

server: https://acme-v02.api.letsencrypt.org/directory

privateKeySecretRef:

The secret that holds the generated private key used to communicate with Let's Encrypt

name: letsencrypt-prod-account-key

dns01:

```
    providers:
    - name: azuredns
      azuredns:
        # Service principal clientId (also called appId)
        clientID: #AZ_CLIENT_ID ← the client id used in the environment variable
        # A secretKeyRef to a service principal ClientSecret (password)
        # ref: https://docs.microsoft.com/en-us/azure/container-service/kubernetes/container-service-kubernetes-service-principal
        clientSecretSecretRef:
          name: azuredns-config
          key: CLIENT_SECRET
        # Azure subscription Id
        subscriptionID: #AZ_SUB_ID ← the subscribtion id used in the environment variable
        tenantID: #AZ_TENANT_ID ← the tenant id used in the environment variable
        resourceGroupName: #AZ_RESOURCE_GROUP(where DNS zone is) ← resource group we used for the newly created domain, in our case it is suseazurecapnoterraform.
        hostedZoneName: #DNS_ZONE_NAME ← the domain name registered by the registrar service, in our case it is suseazurecap.org.
```

Here is the le-prod-cert-issuer.yaml file:

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    # Replace with your email address so you can be notified of expiring certificates
    email: rania.mohamed@suse.com
    # This is using letsencrypt production so beware of rate-limiting!
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      # The secret that holds the generated private key used to communicate with Let's Encrypt
      name: letsencrypt-prod-account-key
    dns01:
```

providers:

- name: azuredns

azuredns:

Service principal clientId (also called appId)

clientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

A secretKeyRef to a service principal ClientSecret (password)

ref: https://docs.microsoft.com/en-us/azure/container-service/kubernetes/container-service-kubernetes-service-principal

clientSecretSecretRef:

name: azuredns-config

key: CLIENT_SECRET

Azure subscription Id

subscriptionID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

tenantID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

resourceGroupName: susecapdemo

hostedZoneName: susecapdemos.com

17. Generate a DST Root CA X3 cert which is now named TrustID X3 Root from https://www.identrust.com/support/downloads by clicking on Base64 Root Certificate under TrustID X3. Capture the certification.
18. Edit "scf-config-values.yaml"

example deployment configuration file

scf-config-values.yaml

env:

the FQDN of your domain

DOMAIN: xxx.xxxxx.xxx ← *a prefix for a subdomain + the newly created domain, in our case I am going to use demo.suseazurecap.org*

the UAA prefix is required

UAA_HOST: uaa.xxxxx.xxxx.xxxx ← *the prefix + the new created domain, demo.suseazurecap.org*

UAA_PORT: 443

UAA_PUBLIC_PORT: 443

GARDEN_ROOTFS_DRIVER: "overlay-xfs"

DEFAULT_STACK: cflinuxfs3

```
  #enable: ← enable this if you want to deploy eirini
  #eirini: true

kube:
  storage_class:
    persistent: "persistent" ← for now the name of the storage class is not configurable, using the terraform scripts.
    shared: "persistent" ← for now the name of the storage class is not configurable, using the terraform scripts.
  auth: "rbac"

  registry:
    hostname: "registry.suse.com"
    username: ""
    password: ""
  organization: "cap"

secrets:
  # Create a very strong password for user 'admin'
  CLUSTER_ADMIN_PASSWORD: xxxxx ← cf and stratos admin password.


  # Create a very strong password, and protect it because it
  # provides root access to everything
  UAA_ADMIN_CLIENT_SECRET: xxxxx ← UAA admin password.

  UAA_CA_CERT: |
    -----BEGIN CERTIFICATE-----
    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ← Replace it with the generated certificate.
    -----END CERTIFICATE-----

    #services:
```

#loadbalanced: true ← *The current support of the terraform scripts only supports ingress and I am going to explain what happens after installing SUSE Cloud Application Platform.*

```
ingress:
  enabled: true
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: 1024m
    certmanager.k8s.io/cluster-issuer: letsencrypt-prod
    certmanager.k8s.io/acme-challenge-type: dns01
    certmanager.k8s.io/acme-dns01-provider: azuredns
```

Here is an example to the scf-config-values.yaml file:

```
### example deployment configuration file
### scf-config-values.yaml

env:
  # the FQDN of your domain
  DOMAIN: demoroon.susecapdemos.com
  # the UAA prefix is required
  UAA_HOST: uaa.demoroon.susecapdemos.com
  UAA_PORT: 443
  UAA_PUBLIC_PORT: 443
  GARDEN_ROOTFS_DRIVER: "overlay-xfs"
  DEFAULT_STACK: cflinuxfs3

  #enable:
  #eirini: true

kube:
  storage_class:
    persistent: "persistent"
    shared: "persistent"
  auth: "rbac"
```

```
 registry:
  hostname: "registry.suse.com"
  username: ""
  password: ""
 organization: "cap"

secrets:
 # Create a very strong password for user 'admin'
 CLUSTER_ADMIN_PASSWORD: xxxxx

 # Create a very strong password, and protect it because it
 # provides root access to everything
 UAA_ADMIN_CLIENT_SECRET: xxxxxxx

 UAA_CA_CERT: |
  -----BEGIN CERTIFICATE-----
  MIIDSjCCAjKgAwIBAgIQRK+wgNajJ7qJMDmGLvhAazANBgkqhkiG9w0BAQUFADA/
  MSQwIgYDVQQKExtEaWdpdGFsIFNpZ25hdHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT
  DkRTVCBSb290IENBIFgzMB4XDTAwMDkzMDIxMTIxOVoXDTIxMDkzMDE0MDExNVow
  PzEkMCIGA1UEChMbRGlnaXRhbCBTaWduYXR1cmUgVHJ1c3QgQ28uMRcwFQYDVQQD
  Ew5EU1QgUm9vdCBDQSBYMzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
  AN+v6ZdQCINXtMxiZfaQguzH0yxrMMpb7NnDfcdAwRgUi+DoM3ZJKuM/IUmTrE4O
  rz5Iy2Xu/NMhD2XSKtkyj4zl93ewEnu1lcCJo6m67XMuegwGMoOifooUMM0RoOEq
  OLl5CjH9UL2AZd+3UWODyOKIYepLYYHsUmu5ouJLGiifSKOeDNoJjj4XLh7dIN9b
  xiqKqy69cK3FCxolkHRyxXtqqzTWMIn/5WgTe1QLyNau7Fqckh49ZLOMxt+/yUFw
  7BZy1SbsOFU5Q9D8/RhcQPGX69Wam40dutolucbY38EVAjqr2m7xPi71XAicPNaD
  aeQQmxkqtilX4+U9m5/wAl0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNV
  HQ8BAf8EBAMCAQYwHQYDVR0OBBYEFMSnsaR7LHH62+FLkHX/xBVghYkQMA0GCSqG
  SIb3DQEBBQUAA4IBAQCjGiybFwBcqR7uKGY3Or+Dxz9LwwmglSBd49lZRNI+DT69
  ikugdB/OEIKcdBodfpga3csTS7MgROSR6cz8faXbauX+5v3gTt23ADq1cEmv8uXr
  AvHRAosZy5Q6XkjEGB5YGV8eAlrwDPGxrancWYaLbumR9YbK+rlmM6pZW87ipxZz
```

R8srzJmwN0jP41ZL9c8PDHIyh8bwRLtTcm1D9SZImlJnt1ir/md2cXjbDaJWFBM5

JDGFoqgCWjBH4d1QB7wCCZAA62RjYJsWvIjJEubSfZGL+T0yjWW06XyxV3bqxbYo

Ob8VZRzI9neWagqNdwvYkQsEjgfbKbYK7p2CNTUQ

-----END CERTIFICATE-----

#services:

#loadbalanced: true

ingress:

enabled: true

annotations:

nginx.ingress.kubernetes.io/proxy-body-size: 1024m

certmanager.k8s.io/cluster-issuer: letsencrypt-prod

certmanager.k8s.io/acme-challenge-type: dns01

certmanager.k8s.io/acme-dns01-provider: azuredns

19. Edit "stratos-metrics-values.yaml":

env:

DOPPLER_PORT: 443

kubernetes:

apiEndpoint: MASTER_URL

prometheus:

kubeStateMetrics:

enabled: true

nginx:

username: xxxx ← *Prometheus user name.*

password: xxxxx ← *Prometheus password.*

services:

loadbalanced: true ←*for now the deployment of Stratos and Metrics can only run using loadbalancer and not following the setup of the core CF and UAA in SUSE Cloud Application Platform*

Here is an example of stratos-metrics-values.yaml file:

env:

```
  DOPPLER_PORT: 443
kubernetes:
  apiEndpoint: MASTER_URL
prometheus:
  kubeStateMetrics:
    enabled: true
nginx:
  username: admin2
  password: xxxxxxxx
services:
  loadbalanced: true
```

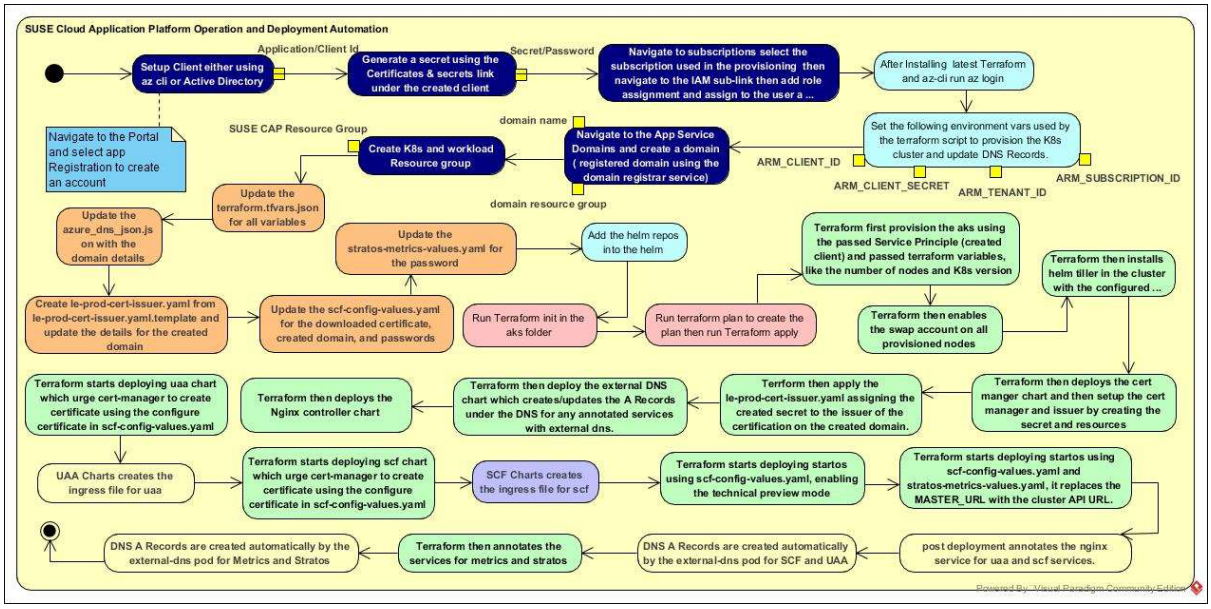
20. Let me explain what will happen when we do init then plan then apply with the Terrraform script:
    a) Using the service principle the Terraform script first provisions the AKS cluster using the configured variables.
    b) Then it enables the swapaccount on the worker nodes before rebooting them
    c) Then it installs helm tiller on the cluster
    d) Then it deploys the D and configures both the certification manager and issuer using the configured values.
    e) Then it deploys the le-prod-cert-issuer. ← at this point you can view the logs of the cert-manager pod to validate that you don't have lock on the DNS or the domain level and to ensure the pasted certification is valid. Note: the certification authority will block the issuing of the certificate after around 5-6 consecutive times, for around a week (+/-), so if you facing a blocking issue then you need to use a new prefix in your scf-config-values.yaml, as the blocking happens on the level of the domain value defined in the scf-config-values.yaml.
    f) Then it deploys the external dns pod which listens for annotations on the services to create/update A Records in the Domain DNS configurations.
    g) Then it deploys Nginx controller chart.
    h) Then it deploys the SUSE Cloud Application Platform charts in the following order:
        i. UAA. Once deployed, the cert manager starts generating its associated certification. You can check the certification status using 'kubectl get certificates -nuaa'
        ii. SCF. Once deployed, the cert manager starts generating its associated certification. You can check the certification status using 'kubectl get certificates -nscf'
        iii. Stratos, which is deployed by default with all preview mode features enabled and using Azure load balancer and not Nginx controller.

iv. Metrics which first generates the configuration file named capConfigurationFile.yaml as a combination of scf-config-values.yaml and stratos-metrics-values.yaml and replacing the MASTER_URL in the later with the Master API Server URL.

v. Once the deployment is done, two ingress are created for UAA and SCF to route all different APIs offered by both components.

vi. The public services created for uaa and scf are annotated with the requested domain prefix so that the external-dns can capture that and create the associated A Records in the DNS configuration of the selected domain.

The following diagram depicts the flow of the deployment using the Terraform scripts:



21. So now let us start Terraform 😊, by running the following commands:

a) First configure Helm by adding all the used repos and checking the version:

i. helm repo add jetstack https://charts.jetstack.io

ii. helm repo add googlestorage https://kubernetes-charts.storage.googleapis.com

iii. helm repo add suse https://kubernetes-charts.suse.com

iv. Check the Helm version. If it is different from 2.14.0 then edit "helm.tf" and change the version.

b) Second, to work around some backward compatibility with Metrics charts:

i. helm fetch suse/metrics

ii. tar -xvzf ./metrics-1.1.0.tgz

iii. edit ./metrics /templates/ config-job.yaml and comment backoffLimit

c) terraform init

d) terraform plan -out ./aks-cap.tfplan

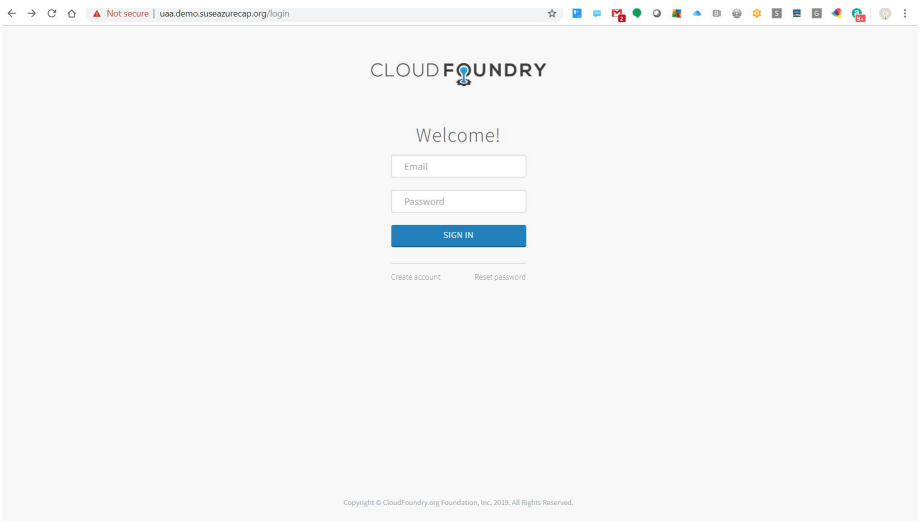
e) terraform apply -input=false ./aks-cap.tfplan ← once you run that you can see the cluster name at the start of creation and you can monitor the deployment once the cluster is created by running:

   i. export KUBECONFIG=./aksk8scfg

   ii. watch --color 'kubectl get po --all-namespaces'

f) Check the certification creation by running the following command 'kubectl get certificate --all-namespaces'. You should view the two certificates ingress-tls for uaa and scf and they should both ready. If not then check the lock on the DNS records and domain level on the portal and also check the logs on the cert-manager pod to find the root cause of the issue.

g) Watch the mysql-0 pod in uaa until it is running it and uaa-0, once they are running validate the following url: https://uaa.demo.suseazurecap.org/. You should see the Uaa Login portal

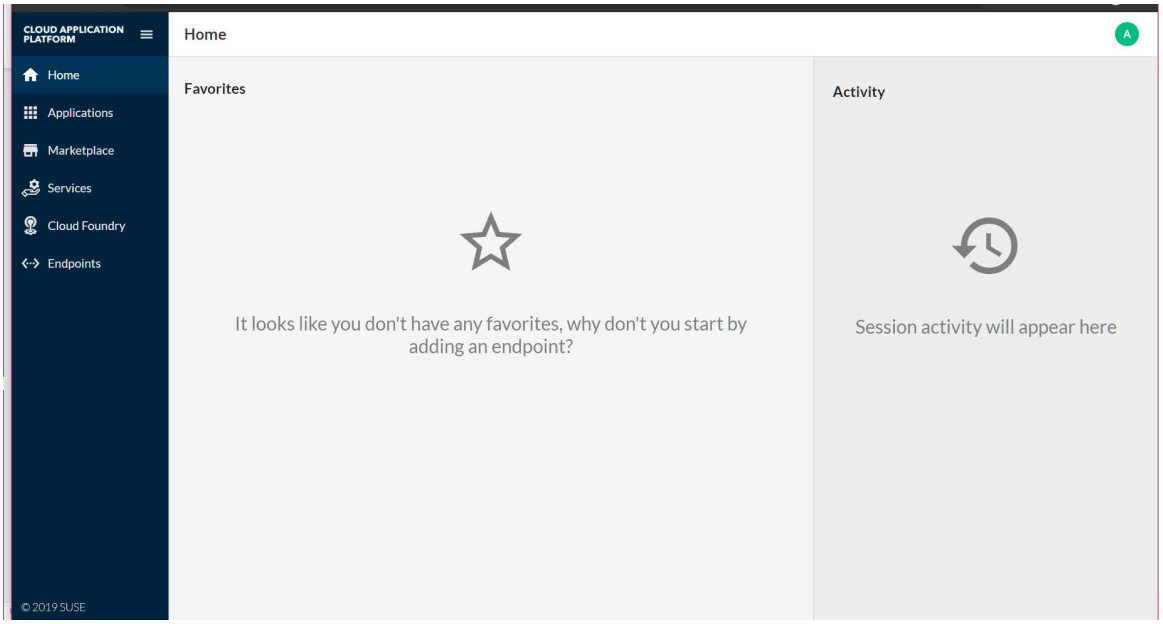


h) Watch all scf pods and wait for them to be ready before checking https://scf.uaa.demo.suseazurecap.org/. You should see the uaa login portal

i) Watch stratos pods and if you see that stratos-0 is stuck at the containerCreating status because of volume mount issues then try deleting all pods 'kubectl delete po stratos-0 stratos-db-6957974946-c4lv5 volume-migration-1-7jgjx -nstratos', replace the name of the db pod accordingly.

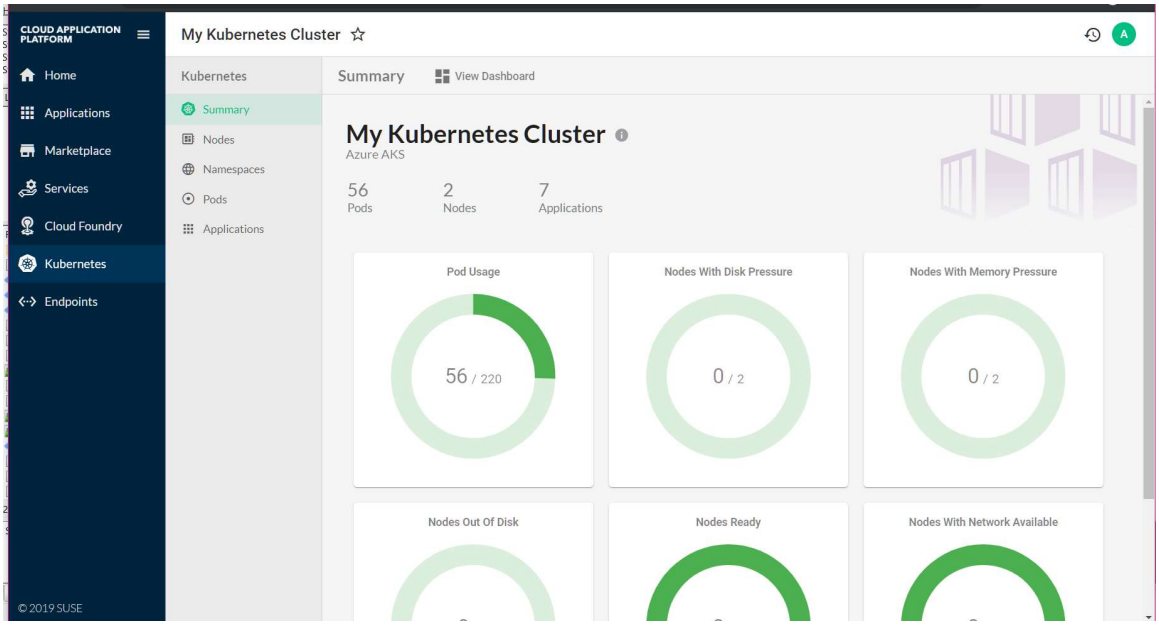
j) Watch metric pods until they are all Running.

k) Now we need to run a script to add the DNS records for stratos and metrics. Now you can login to stratos using https://stratos.demo.suseazurecap.org/ using the admin as username and password in the scf-config-values.yaml.

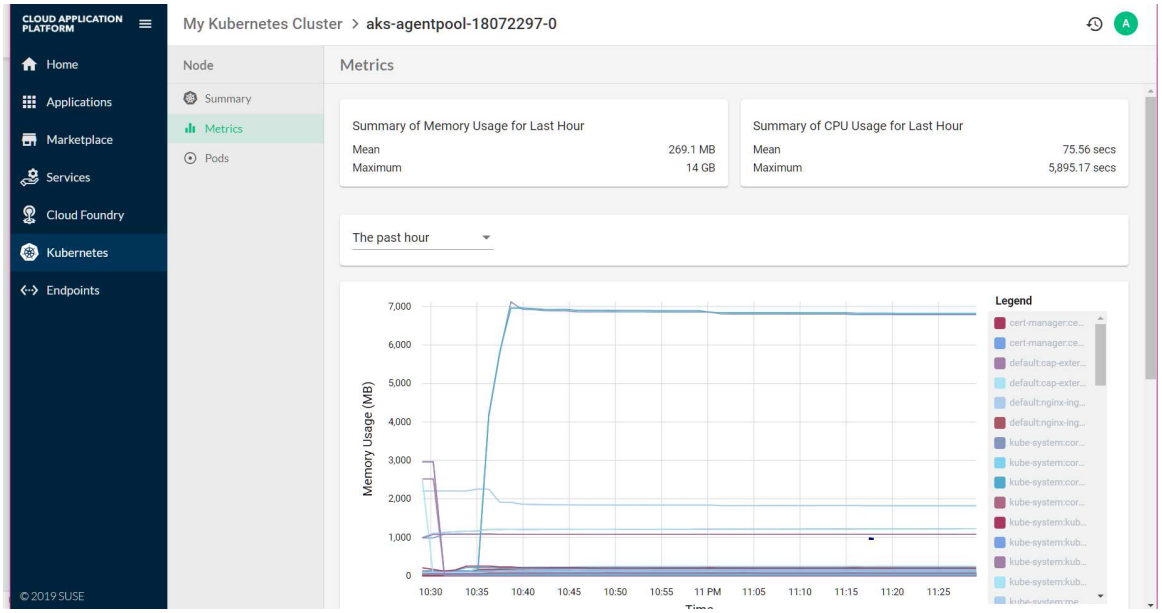


l) Let us now add our Kubernetes and Metrics using stratos. Navigate to Endpoints and add the following:

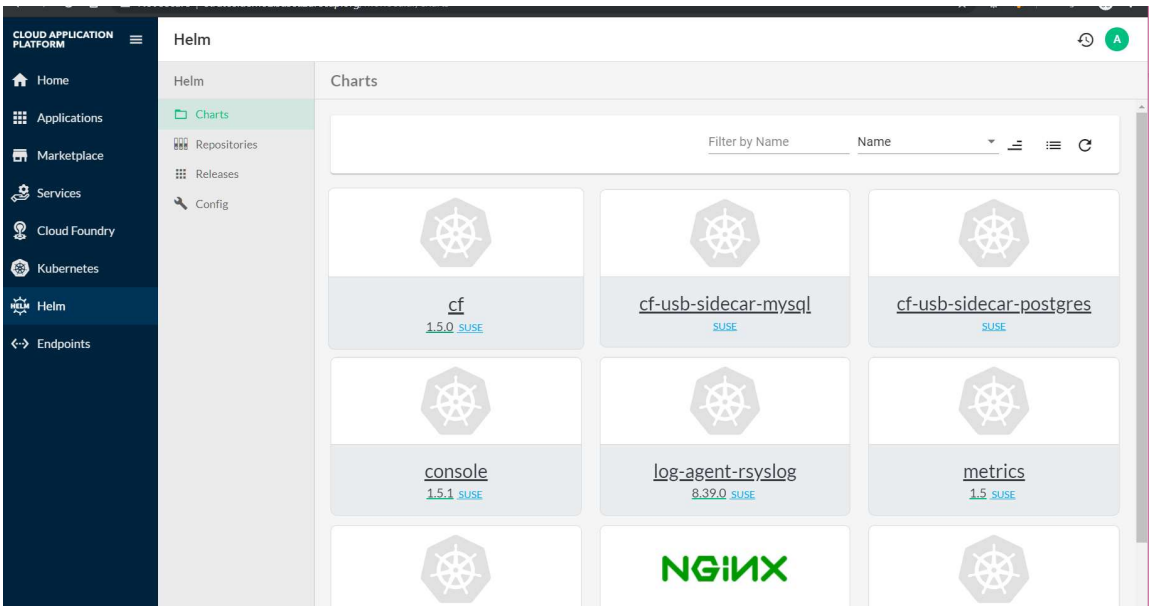
i. Kubernetes: navigate to the endpoint and select add Azure AKS to add the Kubernetes cluster endpoint. Call it My Kubernetes Cluster and get the url of the API server from kubectl cluster-info and use it as the endpoint address. Select skip SSL verification then click on register. Now select connect to the cluster and pass the path to the aksk8scfg file and then click connect and now you will see the Kubernetes endpoint.



ii. Metrics: navigate to the endpoint and select add Prometheus metrics. Call it My Metrics and then enter https://metrics.demo.suseazurecap.org/ in the endpoint. Select skip SSL validation and hit register. Now choose to connect to the endpoint using the username and password configured in stratos-metrics-values.yaml. Now you should see metrics in the Kubernetes nodes and on any deployed application. Here is what the node looks like as we have not yet deployed any application.



iii. Helm: navigate to Endpoints and choose to add a helm repo. I am adding suse repo (https://kubernetes-charts.suse.com) as an example but you may add any helm repo. Make sure to select skip SSL validation before clicking on the finish button. Now you can see the helm repo charts and even deploy any of them in the Kubernetes cluster.



m) Login to SCF using cf cli:

i. cf api https://api.demo.suseazurecap.org

ii. run cf login and enter the email as admin and the password in the scf-config-values.yaml.

iii. now let us create a development space by running cf create-space java-development

iv. now set the system org to the newly created space by running cf target -o "system" -s "java-development"

22. Deploy the Azure service broker and svcat:

    a) First let us install the service broker using the following steps:

        i. export AZURE_SUBSCRIPTION_ID=”xxxxxx” ←*same subscription that we used in the env variable, you may get it again by running ‘az account list -o table’*

        ii. Now let us define some environment var used by the service broker:

            1. export AZURE_TENANT_ID=xxxx ←*same as the one used in the env variable*

            2. export AZURE_CLIENT_ID= xxxx ←*same as the one used in the env variable*

            3. export AZURE_CLIENT_SECRET= xxxx ←*same as the one used in the env variable*

        iii. Install Service Catalog:

            1. helm repo add svc-cat https://svc-catalog-charts.storage.googleapis.com

            2. helm install svc-cat/catalog --name catalog --namespace catalog --set apiserver.storage.etcd.persistence.enabled=true --set apiserver.healthcheck.enabled=false --set controllerManager.healthcheck.enabled=false --set apiserver.verbosity=2 --set controllerManager.verbosity=2

            3. watch for the service catalog pods ‘kubectl get po -ncatalog’

        iv. Install open service broker:

            1. helm repo add azure https://kubernetescharts.blob.core.windows.net/azure

            2. helm install azure/open-service-broker-azure --name osba --namespace osba --set azure.subscriptionId=${AZURE_SUBSCRIPTION_ID} --set azure.tenantId=${AZURE_TENANT_ID} --set azure.clientId=${AZURE_CLIENT_ID} --set azure.clientSecret=${AZURE_CLIENT_SECRET} --set redis.persistence.storageClass=default --set basicAuth.username=$(tr -dc 'a-zA-Z0-9' < /dev/urandom | head -c 16) --set basicAuth.password=$(tr -dc 'a-zA-Z0-9' < /dev/urandom | head -c 16) --set tls.enabled=false

            3. watch for the service broker pods until they are ready ‘kubectl get po -nosba’

    b) Install the svcat:

        i. curl -sLO https://download.svcat.sh/cli/latest/linux/amd64/svcat

        ii. chmod +x ./svcat

        iii. sudo mv ./svcat /usr/local/bin/

        iv. Validate the svact installation by running ‘svcat version –client’

v. Validate the service broker and its associated catalog by running svcat get brokers make sure that it is ready:

```
NAME   NAMESPACE                 URL                                          STATUS
+------+-----------+--------------------------------------------------------+--------+
 osba          http://osba-open-service-broker-azure.osba.svc.cluster.local   Ready
```

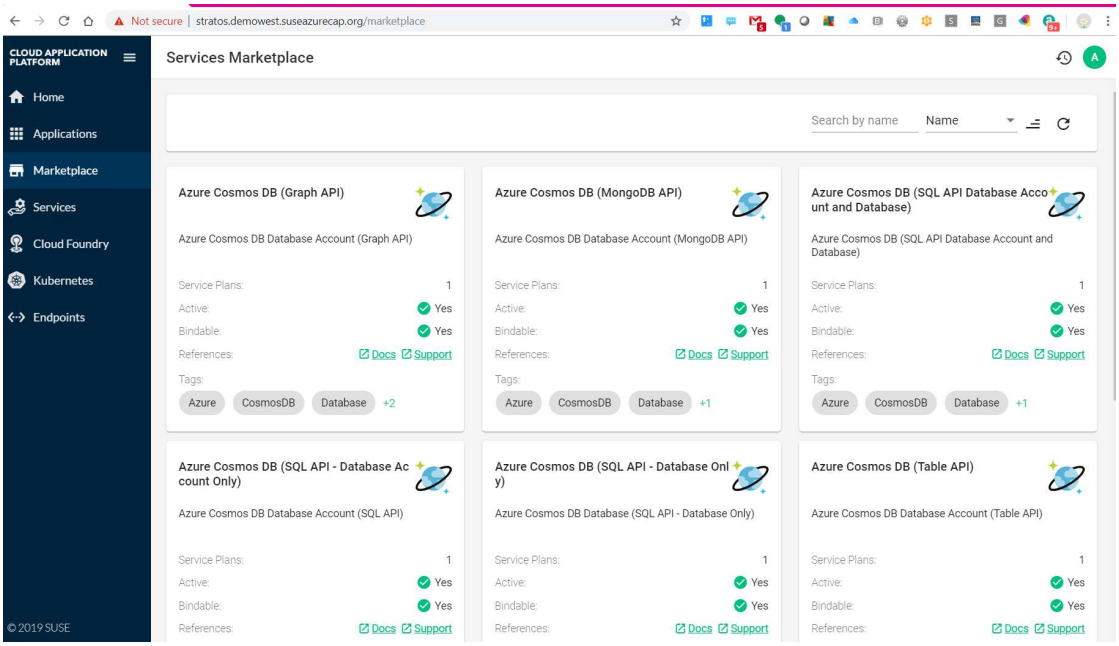
23. Add the service broker to Cloud Foundry by running the following command 'cf create-service-broker azure $(kubectl get deployment osba-open-service-broker-azure --namespace osba --output jsonpath='{.spec.template.spec.containers[0].env[?(@.name == "BASIC_AUTH_USERNAME")].value}') $(kubectl get secret --namespace osba osba-open-service-broker-azure --output jsonpath='{.data.basic-auth-password}' | base64 --decode) http://osba-open-service-broker-azure.osba.svc.cluster.local'

24. Validate the creation by running 'cf service-brokers'. You should see the Azure service broker:

```
name              url
azure             http://osba-open-service-broker-azure.osba.svc.cluster.local
persi-nfs-broker  http://nfs-broker-nfsbroker.scf.svc.cluster.local:8999
```

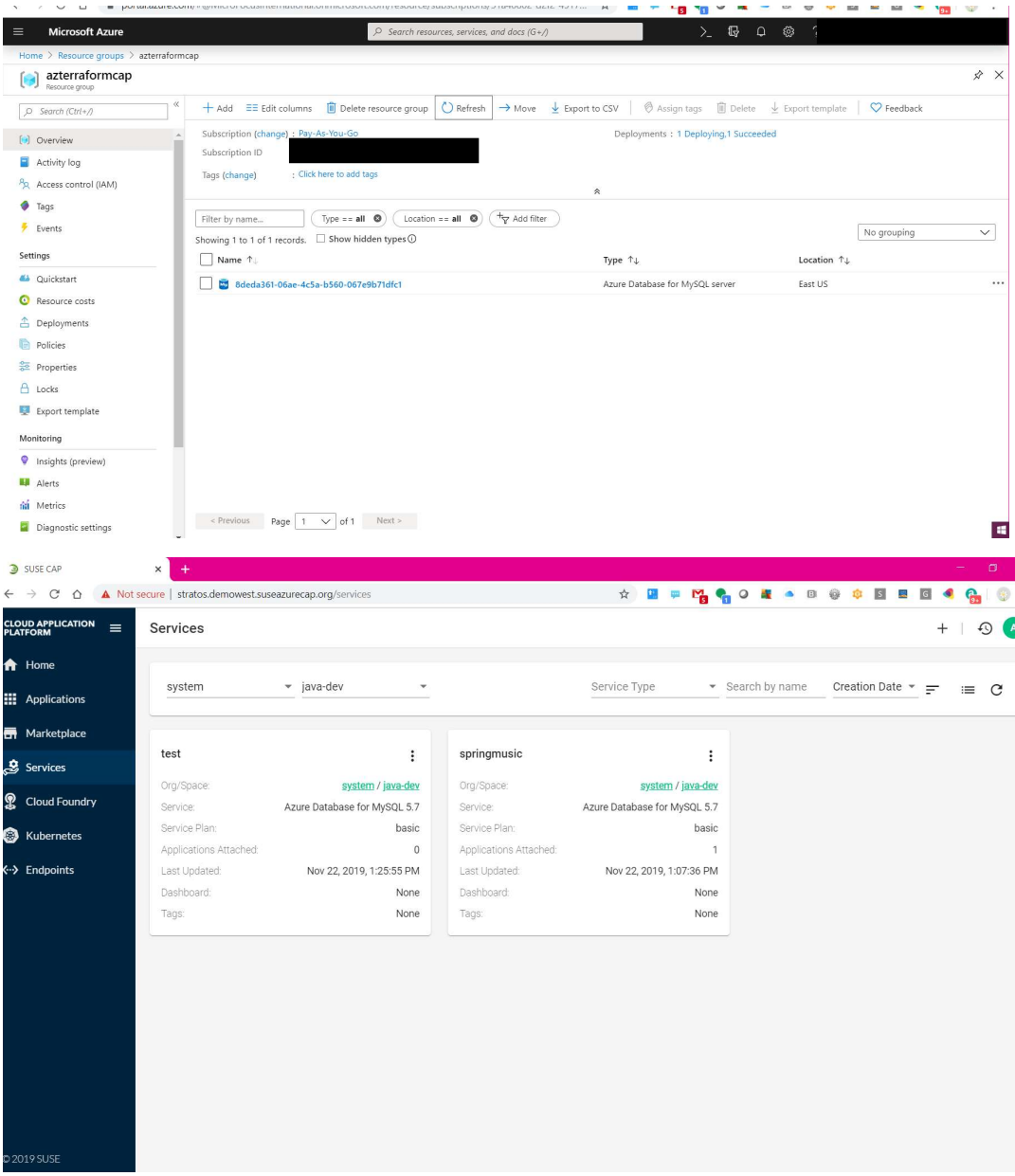
25. Enable some plans for the admin user. I am going to enable all basic plans by running 'cf service-access -b azure | awk '($2 ~ /basic/) { system("cf enable-service-access " $1 " -p " $2)}''

26. You can list all the available services and associated plans by running 'cf service-access -b azure' or view it in the marketplace in Stratos:



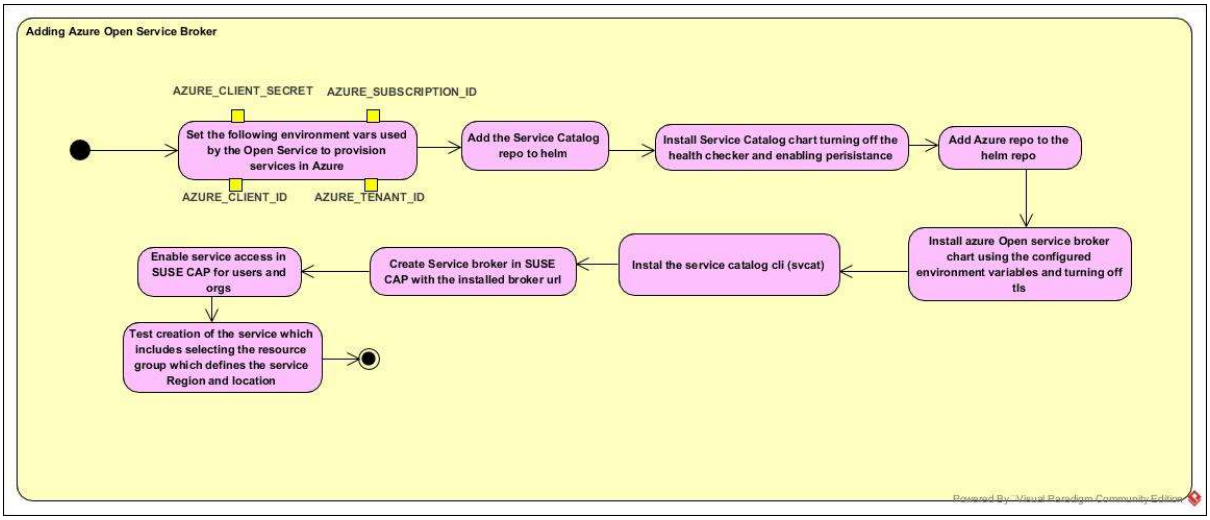
27. Test the service broker by running the following command to create a test database 'cf create-service azure-mysql-5-7 basic test -c "{ \"location\": \"eastus\", \"resourceGroup\": \"azterraformcap\", \"firewallRules\": [{\"name\":\"AllowAll\", \"startIPAddress\":\"0.0.0.0\",\"endIPAddress\":\"255.255.255.255\"}]}"'

28. You can now view the database now in Azure portal and Stratos:



29. Now run ‘cf service test’ to check the status of the created database
30. You may now delete the service using ‘cf purge-service-instance test’ or ‘cf delete-service test’. Use the former if the database on Azure is becomes unavailable for some reason.

    The following diagram depicts the flow of the installation of the service broker:



31. Now let us move to deploying applications (samples) before going to the two java applications 😊:

    a) First application (ruby):

        i. git clone https://github.com/cloudfoundry-samples/ruby-sample-app

        ii. cf push ruby-sample-app

    b) Second application (java Spring):

        i. git clone https://github.com/cloudfoundry-samples/spring-music.git

        ii. install java 8, download it from oracle https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

        iii. run rpm -ivh

        iv. setup JAVA_HOME and PATH variables export JAVA_HOME=/usr/java/jdk1.8.0_231-amd64/ and export PATH=/usr/java/jdk1.8.0_231-amd64/bin:$PATH

        v. run ./gradlew clean assemble

        vi. cf push

        vii. Now let us create a database with the name springmusic:

        cf create-service azure-postgresql-10 basic springmusic -c "{ \"location\": \"eastus\", \"resourceGroup\": \"azterraformcap\", \"firewallRules\": [{\"name\":\"AllowAll\", \"startIPAddress\":\"0.0.0.0\",\"endIPAddress\":\"255.255.255.255\"}]}"

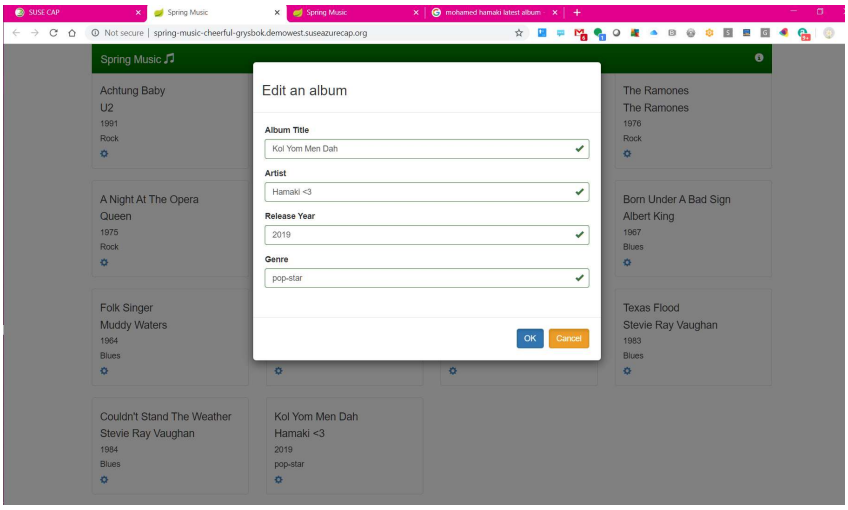
        viii. Watch the service springmusic until is created and updated using 'cf service springmusic'

        ix. Bind it to a service instance 😊 'cf bind-service spring-music springmusic'. Restart the app 'cf restart spring-music'

        x. Now we have an application running with the database 😊.

xi. Now you can test adding a new album and restart the app. Even unbind the database and remove the application and add it back; binding your database again and you will see that the album is still there 😊:

1. cf unbind-service spring-music springmusic
2. cf delete spring-music
3. cf push
4. cf bind-service spring-music springmusic
5. navigate to the application route and you will find the newly added album 😊.



xii. You can now check the metrics of the application for a specific period. It displays the gathered monitoring and application resources consumption details.