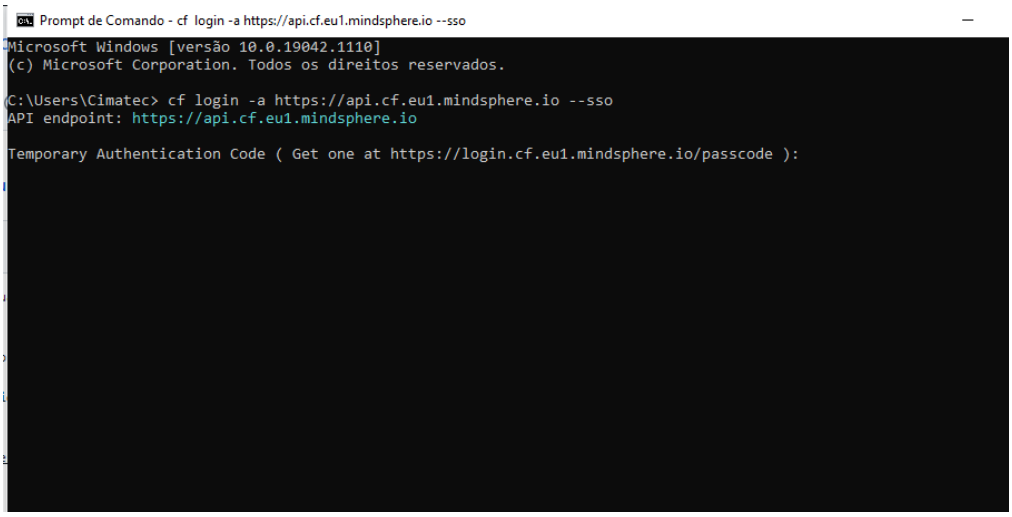
Manual utilização Cloud Foundry (MindSphere)

Sumário

Primeiros passos

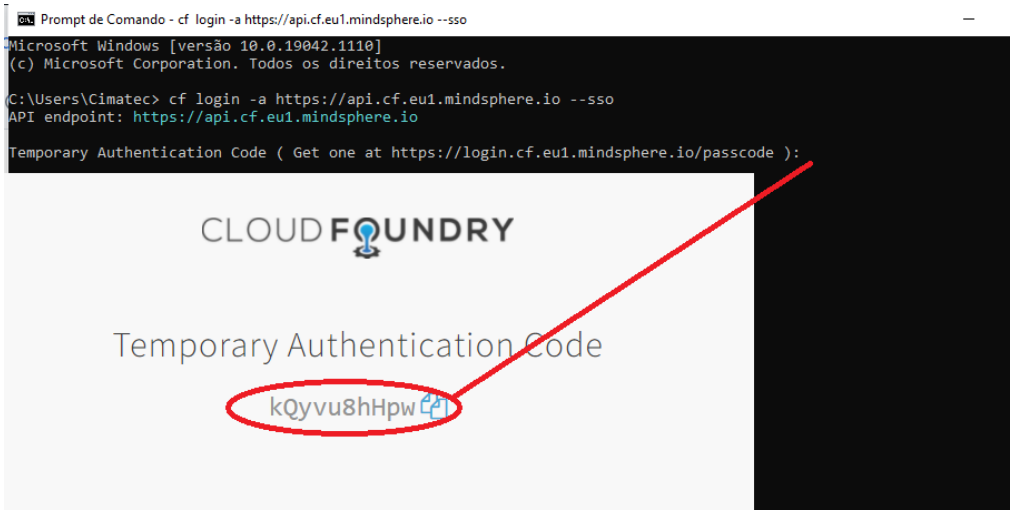
Conectando com o cloud foundry

No terminal utilize o comando `cf login -a https://api.cf.eu1.mindsphere.io --sso`



Se estiver tudo certo vai ser gerado um link para geração de token de autenticação. Esse link deve ser colado no navegador.

Após inserir o código é gerado um token que deve ser adicionado a requisição no terminal.



Tudo certo! Se o código for validado com sucesso você já está autorizado para acessar a plataforma.

Se tudo der certo o retorno será equivalente:

```
API endpoint:   https://api.cf.eu1.mindsphere.io
API version:    3.107.0
user:           SEU USUÁRIO
org:            ORG DA SUA EMPRESA
space:          SPACE DA SUA EMPRESA
```

Configurando conta e/ou ambiente

Caso sua org e/ou space não estejam disponíveis ou não estiver autorizado para fazer o push de uma aplicação

Para poder dar o deploy e gerenciar suas aplicações é preciso que a conta esteja atrelada a um org e space (para mais informações acesse a sessão de referências). Para criar uma Org e Space use os seguintes comandos (Para executar esse comando é preciso ter perfil administrador).

```
cf create-org ORG

cf create-space SPACE [-o ORG] [-q SPACE_QUOTA]
```

Onde "ORG" e "SPACE" é o nome que deseja dar a Org e o Space respectivamente.

Para poder dar o push de uma aplicação é preciso ter a autorização SpaceDeveloper. Para isso use os seguintes comandos:

```
cf set-org-role USERNAME ORG OrgManager
cf set-space-role USERNAME ORG SPACE SpaceDeveloper
```

Onde "USERNAME" é o usuário que deseja atribuir a autorização.

Pronto! Agora o usuário está apto para dar o deploy de uma aplicação.

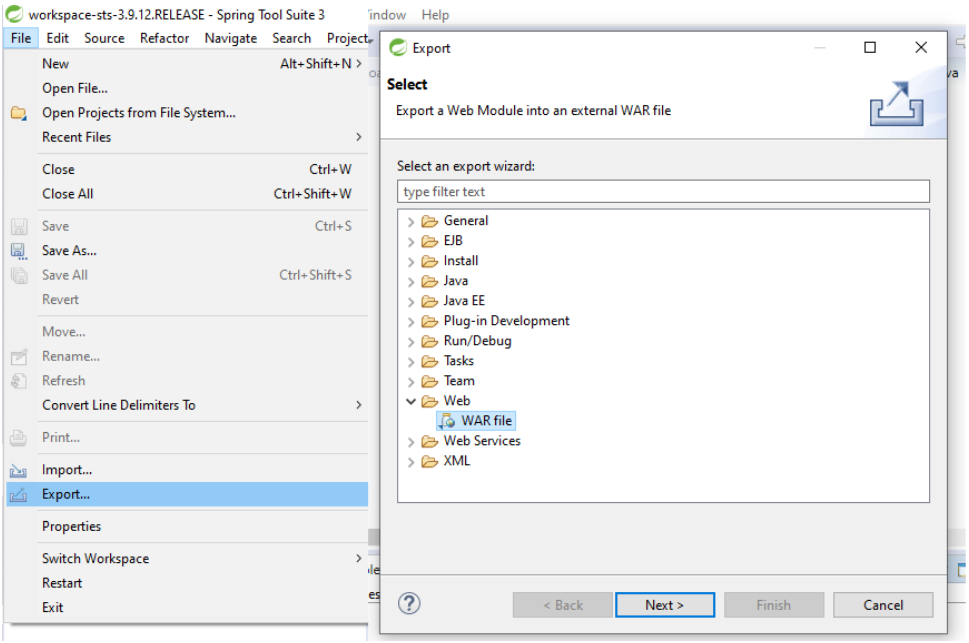
Deploy de aplicação

Antes de continuar assegure-se de ter uma conta com perfil SpaceDeveloper. Utilize o comando `cf space-users ORG SPACE .`

A aplicação utilizada para deploy foi desenvolvida em Spring Boot.

Gerando arquivo .war

Para gerar o arquivo .war, acessar "File->Export-> Selecionar Web -> Clicar em WAR File"



Configuração manifest.ylm

Essa aplicação utiliza o serviço do Postgresql para mais informações visite a sessão Serviços

A configuração básica do manifest.ylm para o projeto é:

```
applications:
  - name: NOMEAPLICAÇÃO
    instances: 1
    buildpacks:
      - java_buildpack
    path: ./APP.war  #Caminho da aplicação
    random-route: true
    memory: 1024MB   #Quantidade de memória ram
    disk_quota: 500MB #Quantidade de espaço disponível para a aplicação
    services:
      - NOMESERVIÇO  #Nome do serviço criado
```

Publicando no Cloud Foundry

Finalizando as configurações basta apenas utilizar o comando `cf push`

Serviços

O serviço utilizado será o do Postgresql

Criando um serviço

Para criar um serviço primeiramente é preciso saber os serviços disponíveis para uso, utilize o seguinte comando:

```
cf marketplace
```

```
Getting all service offerings from marketplace in org cimatec / space cimatec as lucas.jordi@fbter.org.br...

offering         plans                                                                                                  broker                                description
postgresql94     postgresql-m, postgresql-xs, postgresql-m50, postgresql-m150_ha, postgresql-s10, postgresql-s10_ha      a9s-postgresql-broker                 Dedicated PostgreSQL service instances and clusters powered by the anynines Service Framework
rabbitmq36       rabbitmq-m, rabbitmq-xs, rabbitmq-m50, rabbitmq-m150_ha, rabbitmq-s10, rabbitmq-s10_ha                 a9s-rabbitmq-broker                   Dedicated RabbitMQ service instances powered by the anynines Service Framework
logme            logme-m, logme-xs, logme-m150_ha, logme-m50, logme-s10, logme-s10_ha                                    a9s-logme-broker                      Dedicated ELK stacks to monitor applications and service instances powered by the anynines Service Framework
elasticsearch5   elasticsearch-m, elasticsearch-xs, elasticsearch-m50, elasticsearch-m150_ha, elasticsearch-s10, elasticsearch-s10_ha   a9s-elasticsearch-broker   Dedicated Elasticsearch 5.x service instances and clusters powered by the anynines Service Framework
redis32          redis-m, redis-xs, redis-m50, redis-m150_ha, redis-s10, redis-s10_ha                                    a9s-redis-broker                      Dedicated Redis service instances and clusters powered by the anynines Service Framework
redis40          redis-xs, redis-m, redis-m150_ha, redis-m50, redis-s10, redis-s10_ha                                    a9s-redis-broker                      Dedicated Redis service instances and clusters powered by the anynines Service Framework
rabbitmq37       rabbitmq-xs, rabbitmq-m, rabbitmq-m50, rabbitmq-m150_ha, rabbitmq-s10, rabbitmq-s10_ha                  a9s-rabbitmq-broker                   This is a service creating and managing dedicated RabbitMQ service instances, powered by the anynines Service Framework
postgresql10     postgresql-xs, postgresql-m, postgresql-m50, postgresql-m150_ha, postgresql-s10, postgresql-s10_ha      a9s-postgresql-broker                 This is a service creating and managing dedicated PostgreSQL service instances and clusters, powered by the anynines Service Framework
elasticsearch6   elasticsearch-m, elasticsearch-xs, elasticsearch-m50, elasticsearch-m150_ha, elasticsearch-s10, elasticsearch-s10_ha   a9s-elasticsearch-broker   This is the anynines Elasticsearch 6.x service.
postgresql11     postgresql-m, postgresql-xs, postgresql-m50, postgresql-m150_ha, postgresql-s10, postgresql-s10_ha      a9s-postgresql-broker                 This is a service creating and managing dedicated PostgreSQL service instances and clusters, powered by the anynines Service Framework
redis50          redis-m, redis-xs, redis-m50, redis-m150_ha, redis-s10, redis-s10_ha                                    a9s-redis-broker                      This is a service creating and managing dedicated Redis service instances, powered by the anynines Service Framework
rabbitmq38       rabbitmq-m, rabbitmq-m50, rabbitmq-xs, rabbitmq-m150_ha, rabbitmq-s10, rabbitmq-s10_ha                  a9s-rabbitmq-broker                   This is a service creating and managing dedicated Messaging service instances, powered by the anynines Service Framework
```

Para o projeto será utilizado o serviço postgresql94 com o plano postgresql-m. Para criar um serviço no seu Space utilize:

```
cf create-service SERVIÇO PLANO NOMESERVIÇO
```

Para o projeto ficaria o seguinte: `cf create-service postgresql94 postgresql-m postgresql-carsharing`

Antes de utilizar o serviço aguarde alguns minutos até que o status seja atualizado para "create succeeded" para verificar utilize o comando `cf services` e localize o serviço criado.

```
C:\Users\Cimatec>cf services
Getting service instances in org cimatec / space cimatec as lucas.jordi@fbter.org.br...

name                      offering       plan           bound apps    last operation     broker                  upgrade available
postgresql-carsharing     postgresql94   postgresql-m   carsharing    create succeeded   a9s-postgresql-broker   no
postgresql-eletroposto    postgresql94   postgresql-m   eletroposto   update succeeded   a9s-postgresql-broker   no

C:\Users\Cimatec>
```

Criando as chaves do serviço

Para o caso do projeto é preciso criar chaves para acesso ao banco do Postgres:

```
cf create-service-key NOMEDOSERVIÇOCRIADO NOMEDAKEY
```

```
Exemplo :

cf create-service-key postgresql-carsharing cimatec
```

Se a key foi criada com sucesso basta executar o comando:

```
cf service-key NOMEDOSERVIÇO NOMEDAKEY
```

A resposta deve ser algo como:

```
Getting key cimatec for service instance postgresql-carsharing as lucas.jordi@fbter.org.br...

{
  "credentials": {
    "host": "[redacted]",
    "hosts": [
      "[redacted]",
      "[redacted]",
      "[redacted]"
    ],
    "name": "[redacted]",
    "password": "[redacted]",
    "port": 5432,
    "uri": "postgres://[redacted]alias.node.dc1.a9ssvc:5432/
pod6029e1",
    "username": "a9s6418246ae2727820c77476af86a4e8d7dfc976c6"
  }
}
```

Agora é só adicionar o serviço no manifest.yml e as keys na sua aplicação para poder utilizar o serviço.

Acessando serviços utilizando SSH

```
cf ssh -L PORTA:HOSTNAME:PORTADOSERVIÇO NOMEDOAPPQUEUTILIZAOSERVIÇO
```

Digamos que tenha sido criado um serviço chamado serviço1 que é utilizado pela aplicação 1 cadastrada no sistema como app1 cujas keys são:

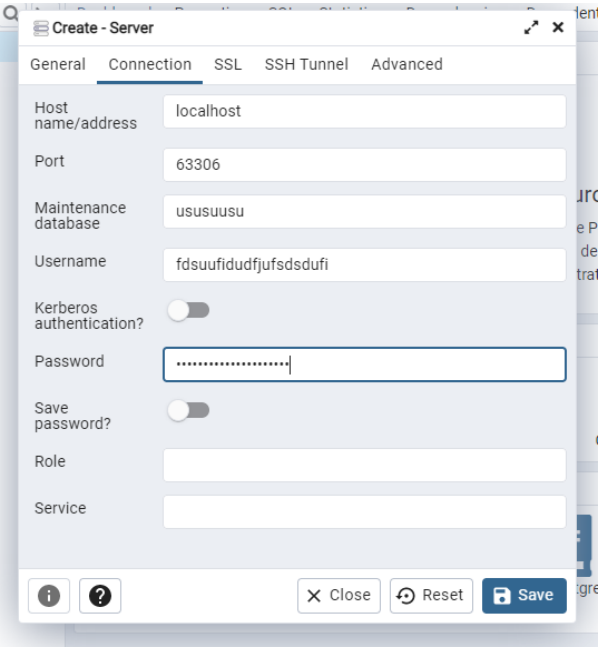
```
{
  "credentials": {
    "host": "ueueue-psql-master-alias.node.dc1.sjusu",
    "hosts": [
      "ueueue-psql-master-alias.node.dc1.djndh",
      "ueueue-psql-master-alias.node.dc1.sjiidiusu",
      "ueueue-psql-master-alias.node.dc1.ddudu"
    ],
    "name": "usususu",
    "password": "viidjfjidfijfdjifd",
    "port": 5432,
    "uri": "postgres://fijsgdsfiogdiogjdoigd:jidsfojofdjisdojifdjoidf@dfiudi-psql-master-alias.node.dc1.a9ssvc:5432/usususu",
    "username": "fdsuufidudfjufsdsdufi"
  }
}
```

Então o comando seria `cf ssh -L 63306:ueueue-psql-master-alias.node.dc1.sjusu:5432 app1`

```
C:\Users\Cimatec>cf ssh -L 63306:pc[redacted]:5432 [redacted]
vcap@f3177993-397c-4f99-78ab-00b0:~$
```

Imagem mostra o resultado da conexão estabelecida.

Agora se quisermos acessar o banco utilizamos o serviço do Postgresql instalado na máquina. Para esse caso foi utilizado o PhpAdmin.



Lembrar de colocar a porta escolhida no SSH e não a que está na key. Nesse caso é a 63306.

É possível ver que as tabelas batem com as do projeto.

Referências

Serviços e Keys

Conectando serviços com SSH

Perfis de autorização

Deploy Spring Apps

Publicando apps