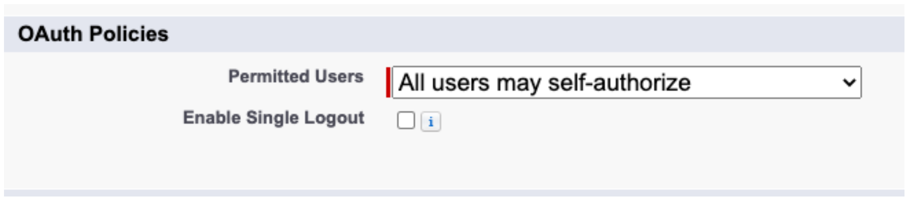
JWT Token flow in Salesforce

Before we begin some important notes on gotchas

1) When you define a connected App that will be used for a JWT token flow then note when you set your policies you cannot allow "All users may self_authorize"



OAuth Policies

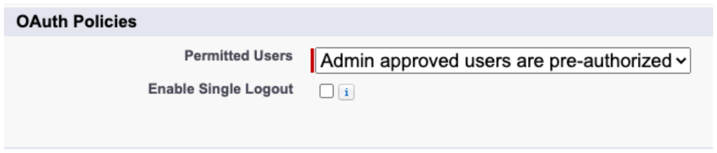
Permitted Users: All users may self-authorize

Enable Single Logout: ☐

```
{"error":"in
```

If you do this then your JWT flow will fail and if you do see and error It will be

```
{"error":"invalid_grant","error_description":"user hasn't approved this consumer"}
```



OAuth Policies

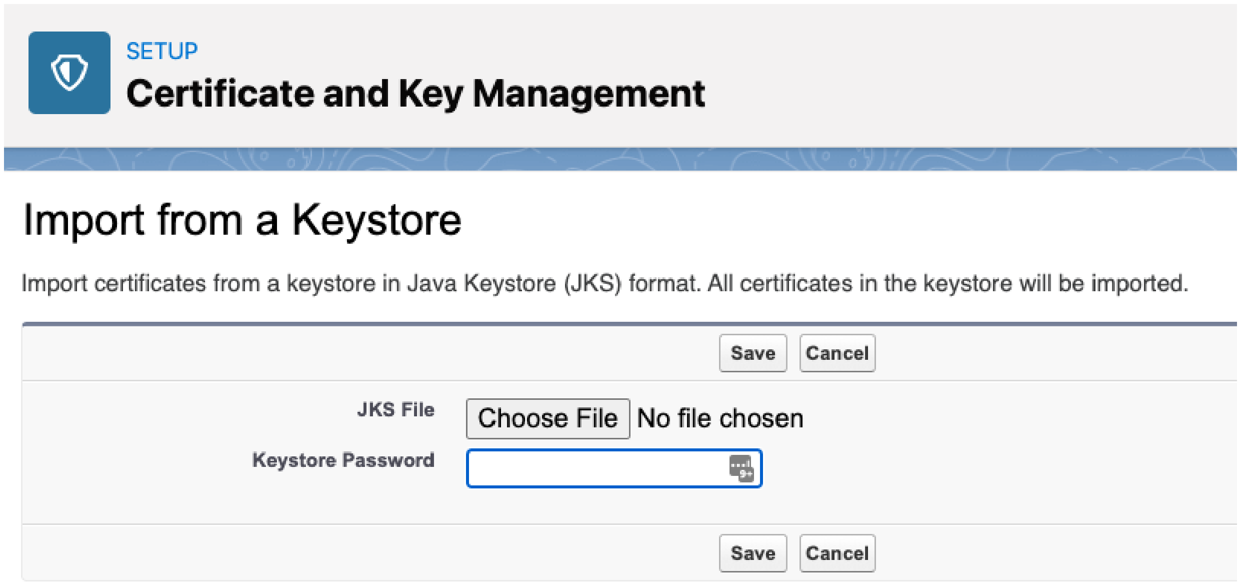
Permitted Users: Admin approved users are pre-authorized

Enable Single Logout: ☐

Make sure that you have selected Admin approved users are pre-Authorized. And then make sure the managed app you have set up is available to the profiles that will be using the JWT connection via the app

2) Later on in this how to guide you are going to need to import a Java Keystore file with your private key / cert in it

more than likely this will not work



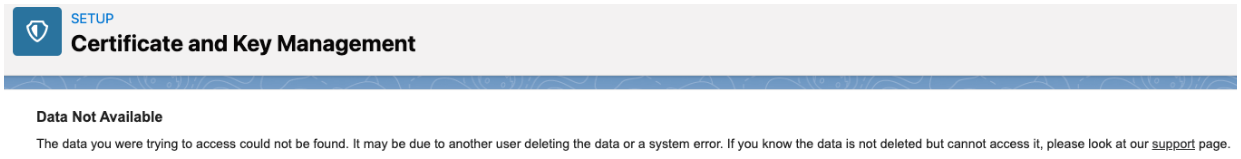
SETUP

Certificate and Key Management

Import from a Keystore

Import certificates from a keystore in Java Keystore (JKS) format. All certificates in the keystore will be imported.

You will go to the Certificate and key management option under security in setup. You will attempt to upload your JKS file. Only to get the following error



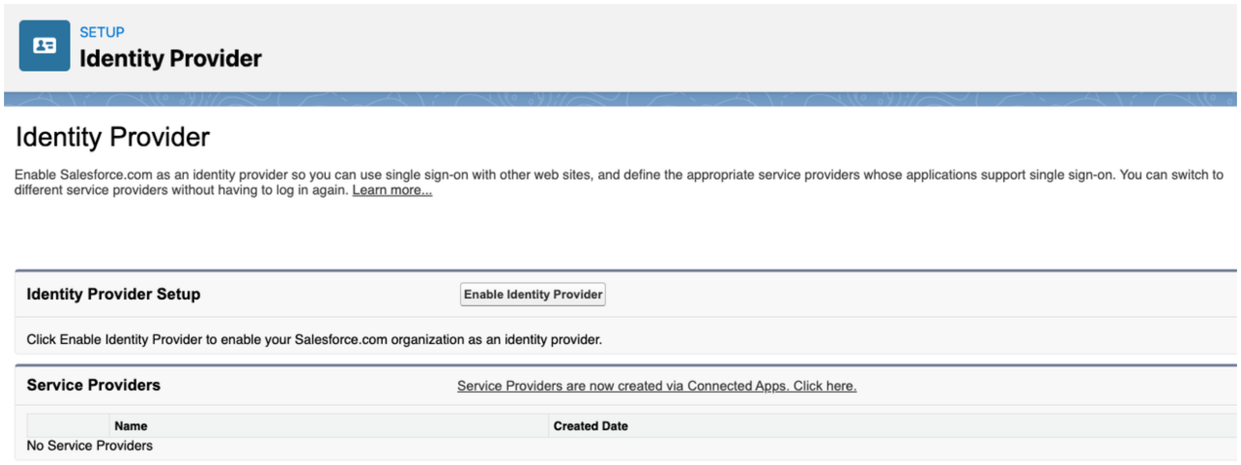
SETUP

Certificate and Key Management

Data Not Available

The data you were trying to access could not be found. It may be due to another user deleting the data or a system error. If you know the data is not deleted but cannot access it, please look at our support page.

There is a super secret way to fix this . You must go to Identity provider in setup and enable it



SETUP

Identity Provider

Identity Provider

Enable Salesforce.com as an identity provider so you can use single sign-on with other web sites, and define the appropriate service providers whose applications support single sign-on. You can switch to different service providers without having to log in again. Learn more...

Identity Provider Setup

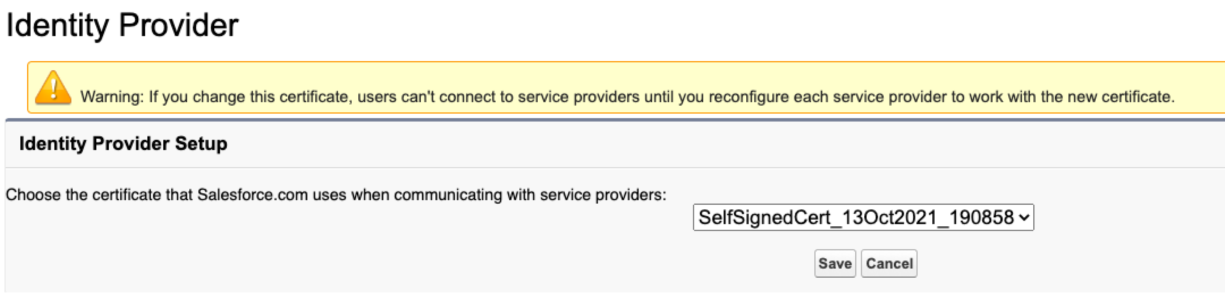
Click Enable Identity Provider to enable your Salesforce.com organization as an identity provider.

Service Providers

Service Providers are now created via Connected Apps. Click here.

Name	Created Date
No Service Providers	

And ignore the scary warnings



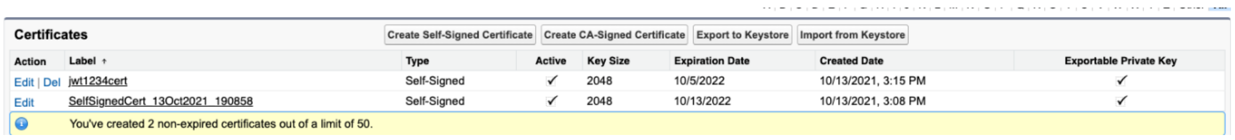
Identity Provider

Warning: If you change this certificate, users can't connect to service providers until you reconfigure each service provider to work with the new certificate.

Identity Provider Setup

Choose the certificate that Salesforce.com uses when communicating with service providers: SelfSignedCert_13Oct2021_190858

Once this has need done got back to Certificates and key management. You will now be able to successfully upload your JKS file to the certificates store jwt1234cert in the screenshot below was mine. This works because REASONS.



Certificates

Action	Label ↑	Type	Active	Key Size	Expiration Date	Created Date	Exportable Private Key
Edit | Del	jwt1234cert	Self-Signed	✓	2048	10/5/2022	10/13/2021, 3:15 PM	✓
Edit	SelfSignedCert_13Oct2021_190858	Self-Signed	✓	2048	10/13/2022	10/13/2021, 3:08 PM	✓

You've created 2 non-expired certificates out of a limit of 50.

You can now go back and disable Identity Provider.

OK NOW WE HAVE COVERED ALL THE GOTCHAS WE CAN GO AHEAD WITH THE INSTRUCTIONS FOR SETTING UP A SALESFORCE TO SALESFORCE CONNECTION VIA JWT TOKEN FLOW. THESE ARE MAC BASED INSTRUCTIONS

We will be using OpenSSl and Keytool to check if these are already installed then type the following into your terminal openssl version

```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl version
LibreSSL 2.8.3
```

If you get a response with a version then you are good

For Keytool just type keytool into the prompt you should get a list of commands if it is installed

```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % keytool
Key and Certificate Management Tool

Commands:

-certreq Generates a certificate request
 -changealias Changes an entry's alias .......................... etc
```

If you have Java installed then Keytool should be installed

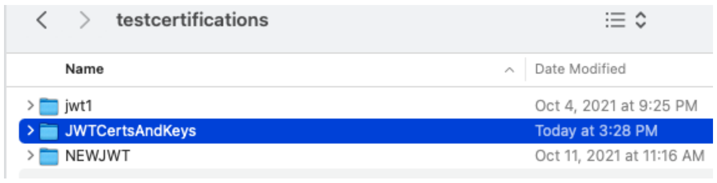
If you do not have openssl then the simplest way to install is via Brew

```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % brew install openssl
```

If you do not have brew (homebrew) installed then WhyNot ????? ok but seriously if you don't then follow the instructions here:-
https://treehouse.github.io/installation-guides/mac/homebrew

Once you have Keytool and Open SSL then let's begin

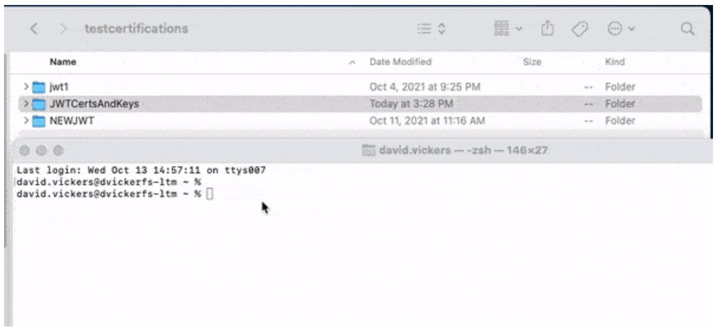
Go to finder and create a folder to store the keys and certs you are about to generate



testcertifications

Name	Date Modified
jwt1	Oct 4, 2021 at 9:25 PM
JWTCertsAndKeys	Today at 3:28 PM
NEWJWT	Oct 11, 2021 at 11:16 AM

Now open a terminal window and drag the folder in



testcertifications

Name	Date Modified	Size	Kind
jwt1	Oct 4, 2021 at 9:25 PM	--	Folder
JWTCertsAndKeys	Today at 3:28 PM	--	Folder
NEWJWT	Oct 11, 2021 at 11:16 AM	--	Folder

david.vickers — -zsh — 146×27

```
Last login: Wed Oct 13 14:57:11 on ttys007
david.vickers@dvickerfs-ltm ~ %
david.vickers@dvickerfs-ltm ~ %
```

Now just use your arrow keys to navigate to the beginning path string and insert cd in front of your path and then hit enter.
example

```
david.vickers@dvickerfs-ltm ~ % cd /Users/david.vickers/Documents/testcertifications/JWTCertsAndKeys
```

This makes the next steps easier as all the files you create will end up in the same directory.
we are going to use the following commands in order

```
openssl genrsa -des3 -passout pass:MyjwtDemo -out server.pass.key 2048
```

```
openssl rsa -passin pass:MyjwtDemo -in server.pass.key -out myserver.key
```

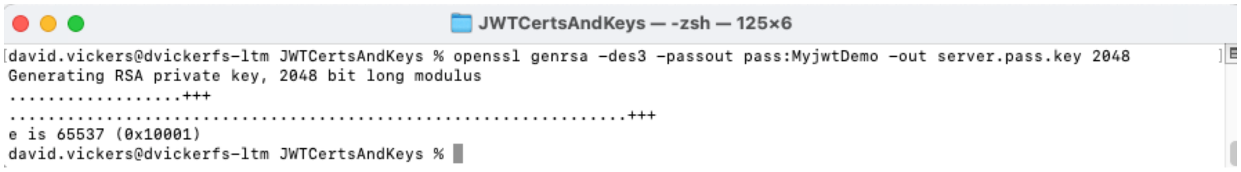
```
openssl req -new -key myserver.key -out myserver.csr
```

```
openssl x509 -req -sha256 -days 365 -in myserver.csr -signkey myserver.key -out myserver.crt
```

At the command Prompt paste in

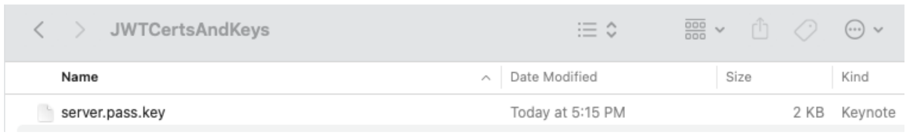
```
openssl genrsa -des3 -passout pass:MyjwtDemo -out server.pass.key 2048
```

Note you are specifying a password (pass:) of `MyjwtDemo`
and creating an output file (-out) called server.pass.key



```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl genrsa -des3 -passout pass:MyjwtDemo -out server.pass.key 2048
Generating RSA private key, 2048 bit long modulus
..................+++
.................................................................+++
e is 65537 (0x10001)
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

you should now have server.pass.key file



Name	Date Modified	Size	Kind
server.pass.key	Today at 5:15 PM	2 KB	Keynote

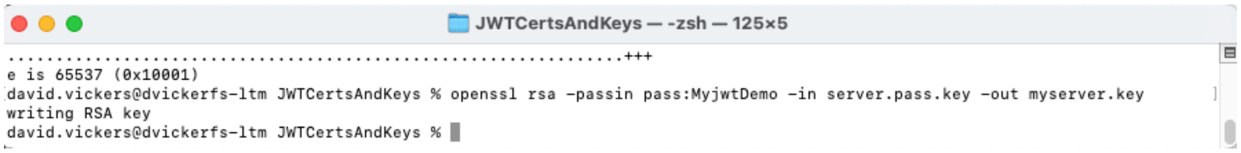
Now insert the next command

openssl rsa -passin pass:MyjwtDemo -in server.pass.key -out myserver.key

note you are passing in the password you set before -passin pass:MyjwtDemo

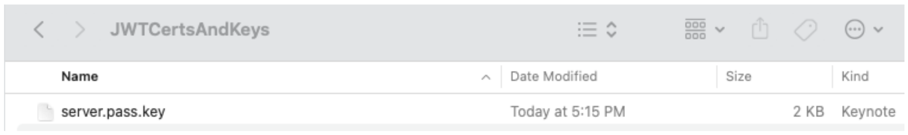
and the input (-in) is your server.pass.key so if you changed that then change it here too!

(-out) you are creating a file called myserver.key



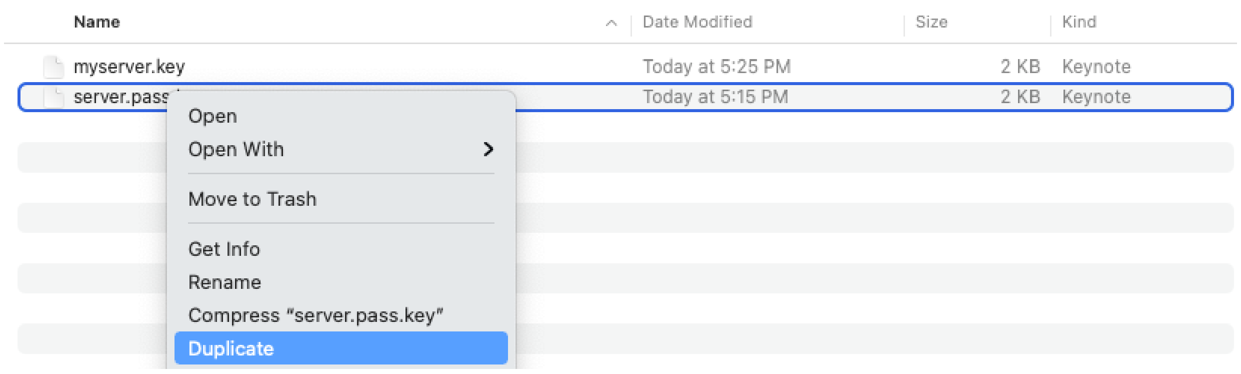
```
.................................................................+++
e is 65537 (0x10001)
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl rsa -passin pass:MyjwtDemo -in server.pass.key -out myserver.key
writing RSA key
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

and you should now have a myserver.key file in your directory



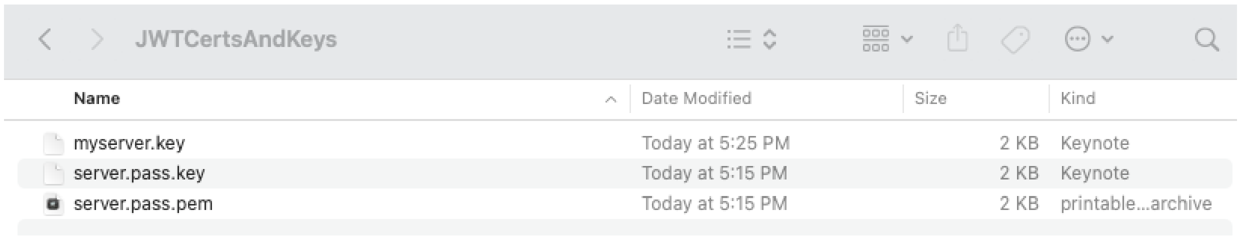
Name	Date Modified	Size	Kind
server.pass.key	Today at 5:15 PM	2 KB	Keynote

Why you are here right click on server.pass.key and duplicate this file



Name	Date Modified	Size	Kind
myserver.key	Today at 5:25 PM	2 KB	Keynote
server.pass	Today at 5:15 PM	2 KB	Keynote

Rename the duplicated of your key file to server.pass.pem. we will need the pem file later!



Name	Date Modified	Size	Kind
myserver.key	Today at 5:25 PM	2 KB	Keynote
server.pass.key	Today at 5:15 PM	2 KB	Keynote
server.pass.pem	Today at 5:15 PM	2 KB	printable...archive

Ok back to the terminal window and paste in the next command

openssl req -new -key myserver.key -out myserver.csr

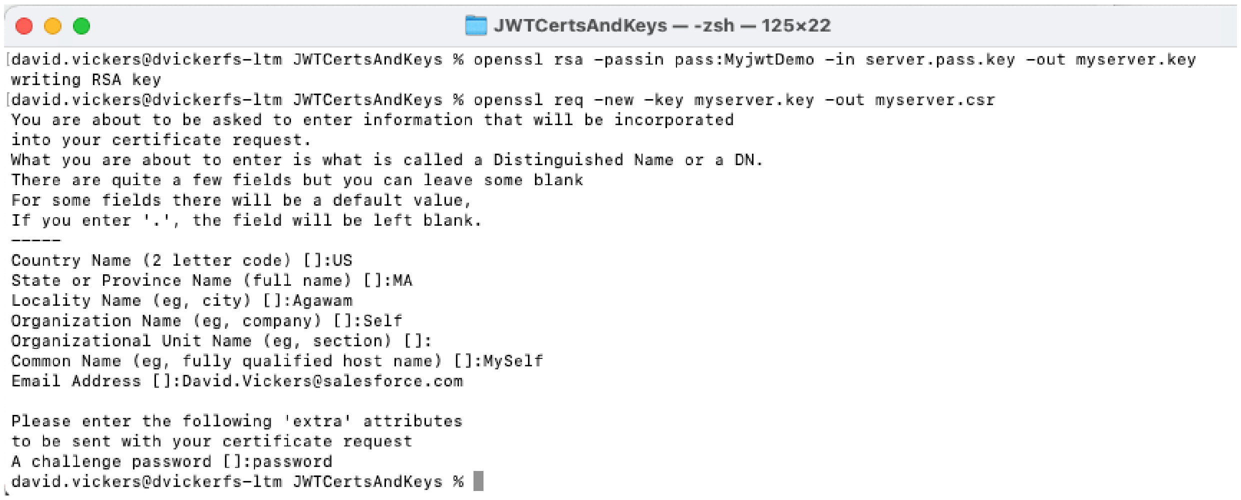
so here we read in the key file and output a csr!

When you enter this you will get questions !

You can fill in as much as you like

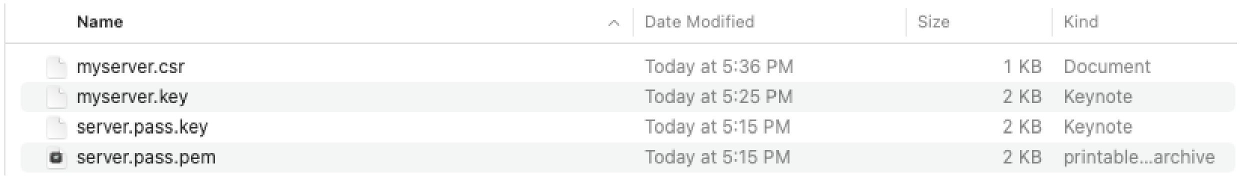
You will also be asked to create a password I suggest for the next steps keeping the same password as it's much easier to keep track off. This password protects you certificate so you need to remember it

I will use the cunning password .....password



```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl rsa -passin pass:MyjwtDemo -in server.pass.key -out myserver.key
writing RSA key
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl req -new -key myserver.key -out myserver.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:US
State or Province Name (full name) []:MA
Locality Name (eg, city) []:Agawam
Organization Name (eg, company) []:Self
Organizational Unit Name (eg, section) []:
Common Name (eg, fully qualified host name) []:MySelf
Email Address []:David.Vickers@salesforce.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

Check your Directory again and you will see you have a CSR file.

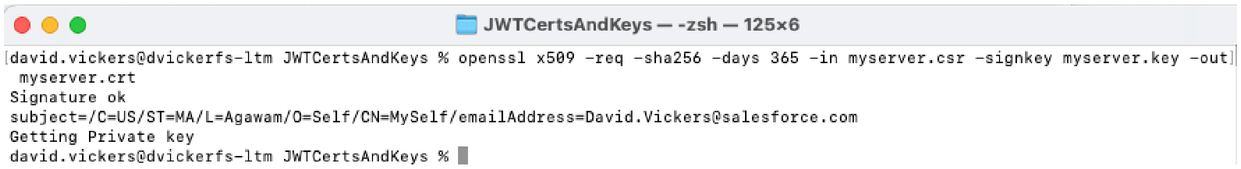


Name	Date Modified	Size	Kind
myserver.csr	Today at 5:36 PM	1 KB	Document
myserver.key	Today at 5:25 PM	2 KB	Keynote
server.pass.key	Today at 5:15 PM	2 KB	Keynote
server.pass.pem	Today at 5:15 PM	2 KB	printable...archive

We need a 509 certificate so now we can use our csr file to create that
paste in the next command

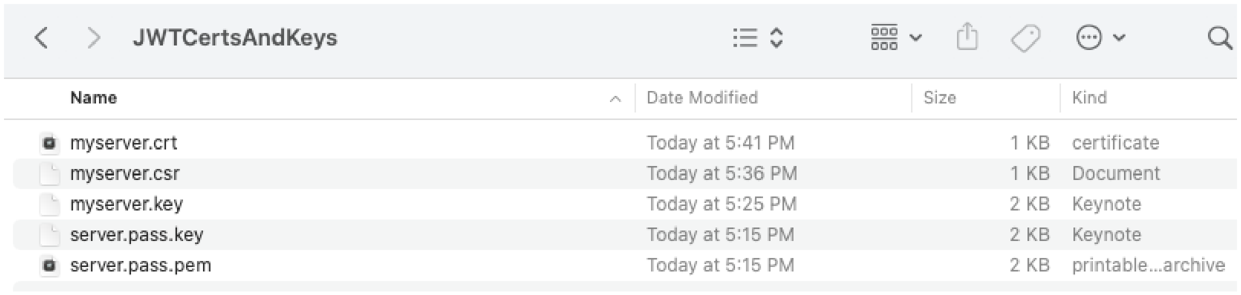
openssl x509 -req -sha256 -days 365 -in myserver.csr -signkey myserver.key -out myserver.crt

note this reads in both your CSR and your signing (private key) that you created earlier and outputs a cert valid for 365 days (max for a sha256 cert)



```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % openssl x509 -req -sha256 -days 365 -in myserver.csr -signkey myserver.key -out
 myserver.crt
Signature ok
subject=/C=US/ST=MA/L=Agawam/O=Self/CN=MySelf/emailAddress=David.Vickers@salesforce.com
Getting Private key
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

And with that you will have a valid certificate in your directory



Name	Date Modified	Size	Kind
myserver.crt	Today at 5:41 PM	1 KB	certificate
myserver.csr	Today at 5:36 PM	1 KB	Document
myserver.key	Today at 5:25 PM	2 KB	Keynote
server.pass.key	Today at 5:15 PM	2 KB	Keynote
server.pass.pem	Today at 5:15 PM	2 KB	printable...archive

Ok for creating a connected app to support a JWT token flow this is all you need, but for a Salesforce to Salesforce connection we are going to be using Named Credentials and for that we are going to need to store our certificate in Salesforce and for that we need this in a Java Key store format (jks)

So onward!!! here are the next commands we will use

Openssl pkcs12 -export -in myserver.crt -inkey server.pass.pem -out mykeystore.p12

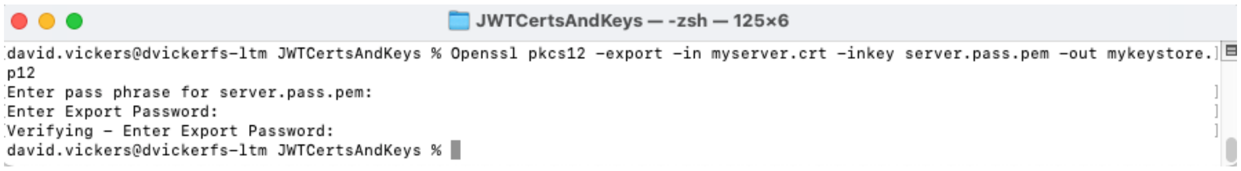
Keytool -importkeystore -srckeystore mykeystore.p12 -srcstoretype pkcs12 -destkeystore servercert.jks -deststoretype JKS

Keytool -keystore servercert.jks -changealias -alias 1 -destalias jwtDemo

First we paste into the terminal window
Openssl pkcs12 -export -in myserver.crt -inkey server.pass.pem -out mykeystore.p12
Note this uses that PEM file we created earlier (the copy of the key file)
This routing takes in the cert and the pem file and outputs a keystore in a p12 format you will need to input your PEM password, so if you forgot that then you are now SOL! but this was the password you used to generate the key remember

openssl genrsa -des3 -passout pass:MyjwtDemo -out server.pass.key 2048
Note you are specifying a password (pass:) of MyjwtDemo

If you get this right then your terminal window will look as below and ask for an export password and to verify that



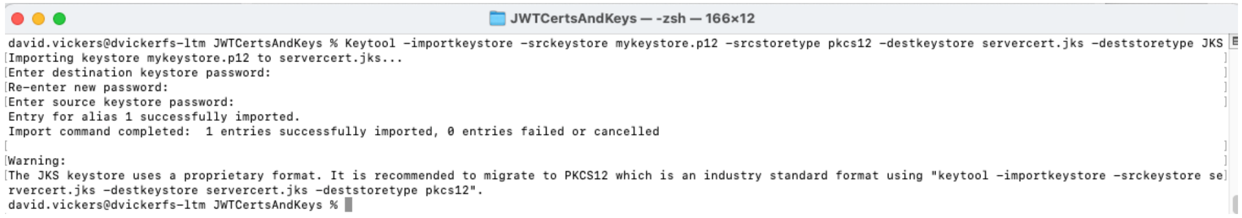
```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % Openssl pkcs12 -export -in myserver.crt -inkey server.pass.pem -out mykeystore.
p12
Enter pass phrase for server.pass.pem:
Enter Export Password:
Verifying - Enter Export Password:
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

and you will get a 2kb mykeystore.p12 file. if you get a zero kb file then the operation failed delete the file and try again

So now with your p12 file you need to make it a .jks file so we run the next command

```
Keytool -importkeystore -srckeystore mykeystore.p12 -srcstoretype pkcs12 -destkeystore servercert.jks -deststoretype JKS
```

we will be prompted for a password. this is the password you set for the export password in the previous step you will then be asked for a destination password you will begin to see that there are a lot of passwords in this sequence so staying with the same one all the way though makes things simpler! Your terminal window will look like this



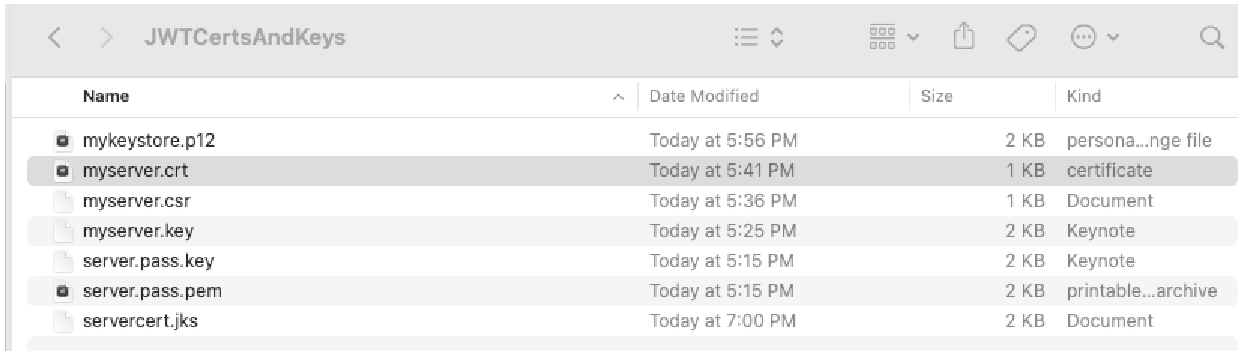
```
david.vickers@dvickerfs-ltm JWTCertsAndKeys % Keytool -importkeystore -srckeystore mykeystore.p12 -srcstoretype pkcs12 -destkeystore servercert.jks -deststoretype JKS
Importing keystore mykeystore.p12 to servercert.jks...
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeystore -srckeystore se
rvercert.jks -destkeystore servercert.jks -deststoretype pkcs12".
david.vickers@dvickerfs-ltm JWTCertsAndKeys %
```

don’t worry about the warning

Okay we are nearly done there is one final issue this PKS file will have the default alias of 1. Salesforce does not allow you to upload a JKS file with the default alias so we need to change it. The Alias you give will be the label of the Certificate / keystore entry when you upload it to Salesforce.
Paste in the final command

```
Keytool -keystore servercert.jks -changealias -alias 1 -destalias jwtDemo
```

Enter your password and hit enter. you will get the same warning as before but your JKS file will now have a new alias in this case “jwtdemo” (unless you picked a different alias)
Your directory now should look like this



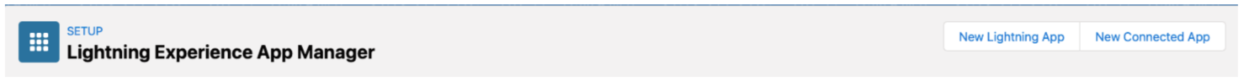
Name	Date Modified	Size	Kind
mykeystore.p12	Today at 5:56 PM	2 KB	persona...nge file
myserver.crt	Today at 5:41 PM	1 KB	certificate
myserver.csr	Today at 5:36 PM	1 KB	Document
myserver.key	Today at 5:25 PM	2 KB	Keynote
server.pass.key	Today at 5:15 PM	2 KB	Keynote
server.pass.pem	Today at 5:15 PM	2 KB	printable...archive
servercert.jks	Today at 7:00 PM	2 KB	Document

you now have all the files you will need!

Setting Up your Target Salesforce Org

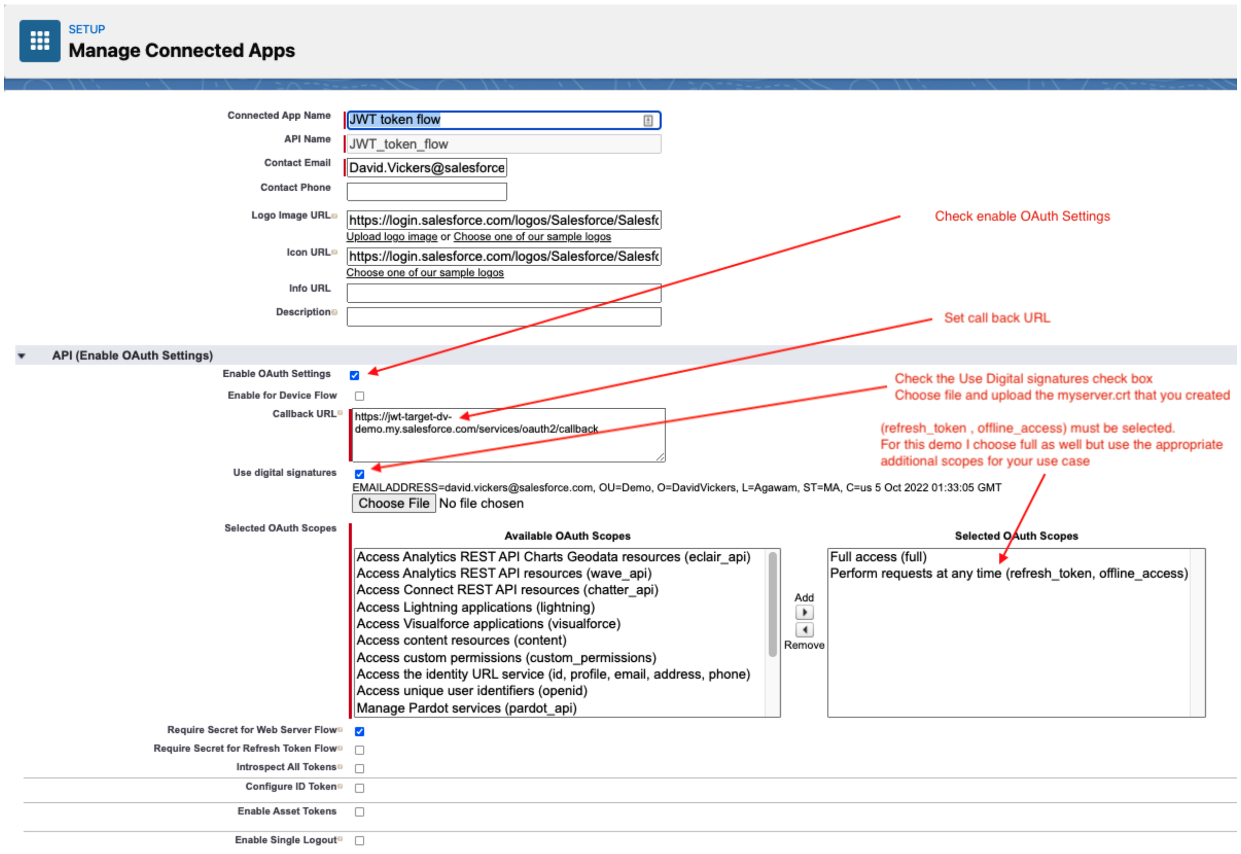
Your Target org is going to need a connected app that will be used to make the connection into

Setup
Apps → app manager



Click New connected app

Fill in the connected app page in a similar fashion to that shown below. key details to pay attention to are indicated via the red arrows


SETUP
Manage Connected Apps

Connected App Name	JWT token flow
API Name	JWT_token_flow
Contact Email	David.Vickers@salesforce
Contact Phone	
Logo Image URL	https://login.salesforce.com/logos/Salesforce/Salesfc
	Upload logo image or Choose one of our sample logos
Icon URL	https://login.salesforce.com/logos/Salesforce/Salesfc
	Choose one of our sample logos
Info URL	
Description	

API (Enable OAuth Settings)

Enable OAuth Settings	✓
Enable for Device Flow	☐
Callback URL	https://jwt-target-dv-demo.my.salesforce.com/services/oauth2/callback
Use digital signatures	✓
	EMAILADDRESS=david.vickers@salesforce.com, OU=Demo, O=DavidVickers, L=Agawam, ST=MA, C=us 5 Oct 2022 01:33:05 GMT
	Choose File No file chosen

Selected OAuth Scopes

Available OAuth Scopes:
- Access Analytics REST API Charts Geodata resources (eclair_api)
- Access Analytics REST API resources (wave_api)
- Access Connect REST API resources (chatter_api)
- Access Lightning applications (lightning)
- Access Visualforce applications (visualforce)
- Access content resources (content)
- Access custom permissions (custom_permissions)
- Access the identity URL service (id, profile, email, address, phone)
- Access unique user identifiers (openid)
- Manage Pardot services (pardot_api)

Selected OAuth Scopes:
- Full access (full)
- Perform requests at any time (refresh_token, offline_access)

Require Secret for Web Server Flow	✓
Require Secret for Refresh Token Flow	☐
Introspect All Tokens	☐
Configure ID Token	☐
Enable Asset Tokens	☐
Enable Single Logout	☐

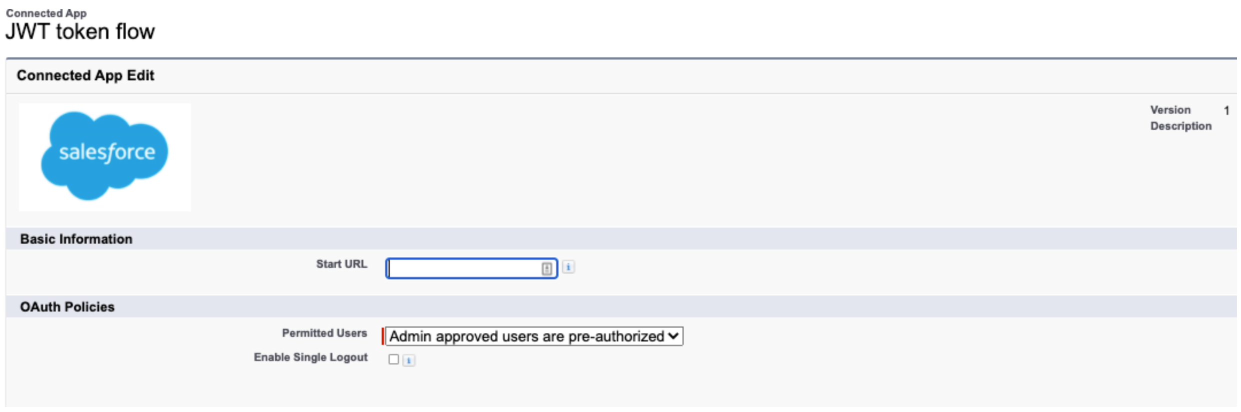
Check enable OAuth Settings

Set call back URL

Check the Use Digital signatures check box
Choose file and upload the myserver.crt that you created

(refresh_token , offline_access) must be selected.
For this demo I choose full as well but use the appropriate additional scopes for your use case

Save your connected app
Now click on Manage to manage you connected app and Edit Policies


Connected App
JWT token flow

Connected App Edit

Version	1
Description	

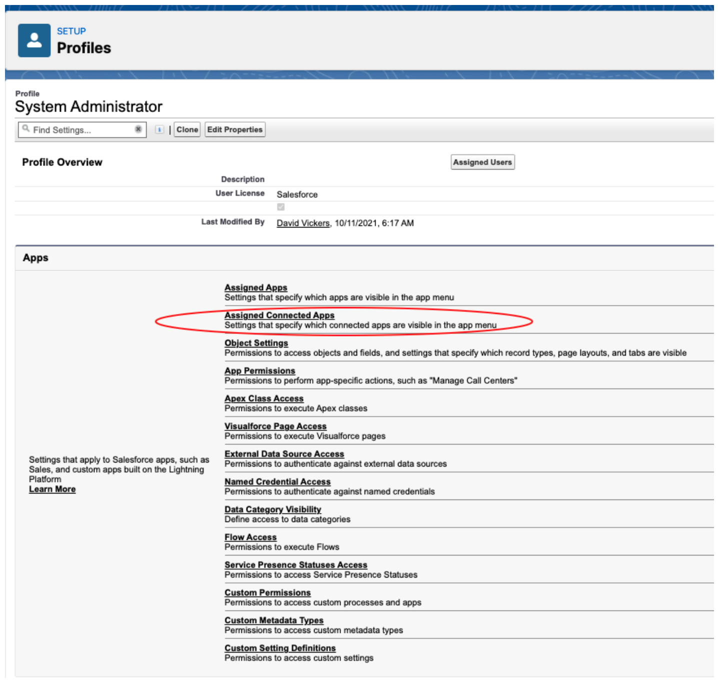
Basic Information

Start URL	

OAuth Policies

Permitted Users	Admin approved users are pre-authorized
Enable Single Logout	☐

Make sure to set permitted users to “admin approved users are re-authorized
Go to the profile / profiles that you want to be able to connect to SF via this connected app
Select assigned connected apps.


SETUP
Profiles

Profile
System Administrator

Find Settings... | Clone | Edit Properties

Profile Overview Assigned Users

Description	
User License	Salesforce
	☐
Last Modified By	David Vickers, 10/11/2021, 6:17 AM

Apps

Settings that apply to Salesforce apps, such as Sales, and custom apps built on the Lightning Platform
Learn More

Assigned Apps
Settings that specify which apps are visible in the app menu

Assigned Connected Apps
Settings that specify which connected apps are visible in the app menu

Object Settings
Permissions to access objects and fields, and settings that specify which record types, page layouts, and tabs are visible

App Permissions
Permissions to perform app-specific actions, such as "Manage Call Centers"

Apex Class Access
Permissions to execute Apex classes

Visualforce Page Access
Permissions to execute Visualforce pages

External Data Source Access
Permissions to authenticate against external data sources

Named Credential Access
Permissions to authenticate against named credentials

Data Category Visibility
Define access to data categories

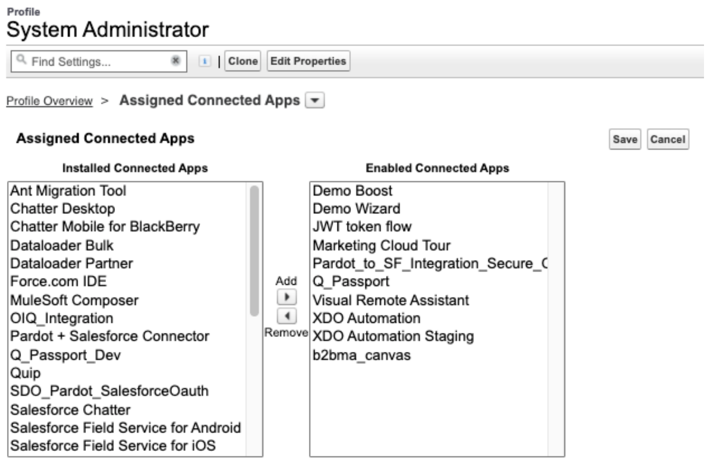
Flow Access
Permissions to execute Flows

Service Presence Statuses Access
Permissions to access Service Presence Statuses

Custom Permissions
Permissions to access custom processes and apps

Custom Metadata Types
Permissions to access custom metadata types

Custom Setting Definitions
Permissions to access custom settings


Profile
System Administrator

Find Settings... | Clone | Edit Properties

Profile Overview > **Assigned Connected Apps**

Assigned Connected Apps Save Cancel

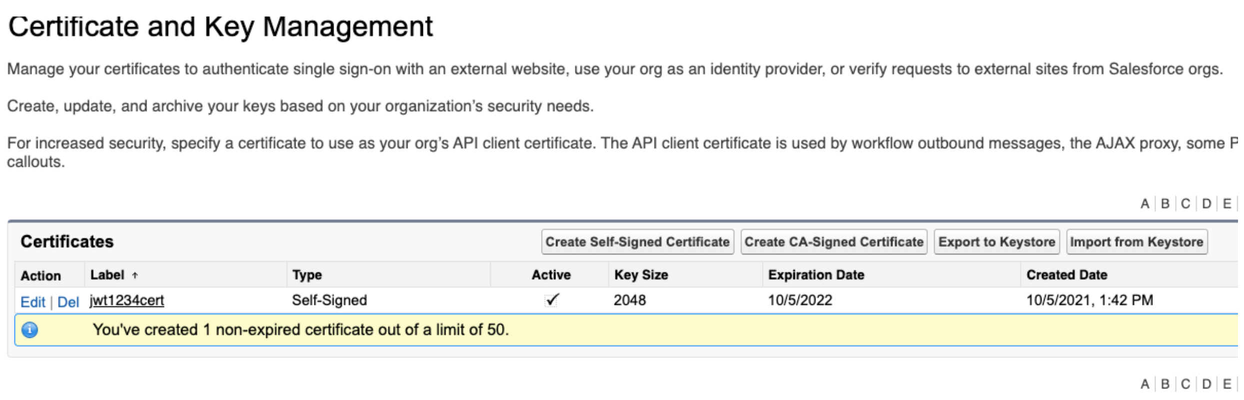
Installed Connected Apps		Enabled Connected Apps
Ant Migration Tool		Demo Boost
Chatter Desktop		Demo Wizard
Chatter Mobile for BlackBerry		JWT token flow
Dataloader Bulk		Marketing Cloud Tour
Dataloader Partner		Pardot_to_SF_Integration_Secure_C
Force.com IDE	Add	Q_Passport
MuleSoft Composer		Visual Remote Assistant
OIQ_Integration		XDO Automation
Pardot + Salesforce Connector	Remove	XDO Automation Staging
Q_Passport_Dev		b2bma_canvas
Quip		
SDO_Pardot_SalesforceOauth		
Salesforce Chatter		
Salesforce Field Service for Android		
Salesforce Field Service for iOS		

Select the connected app you just created (JWT token flow in my case) and move it into the enabled connected apps and save.

You are now done with this ORG

Now To Set the sending org

Got to Setup → Security → Certificate and Key Management


Certificate and Key Management

Manage your certificates to authenticate single sign-on with an external website, use your org as an identity provider, or verify requests to external sites from Salesforce orgs.

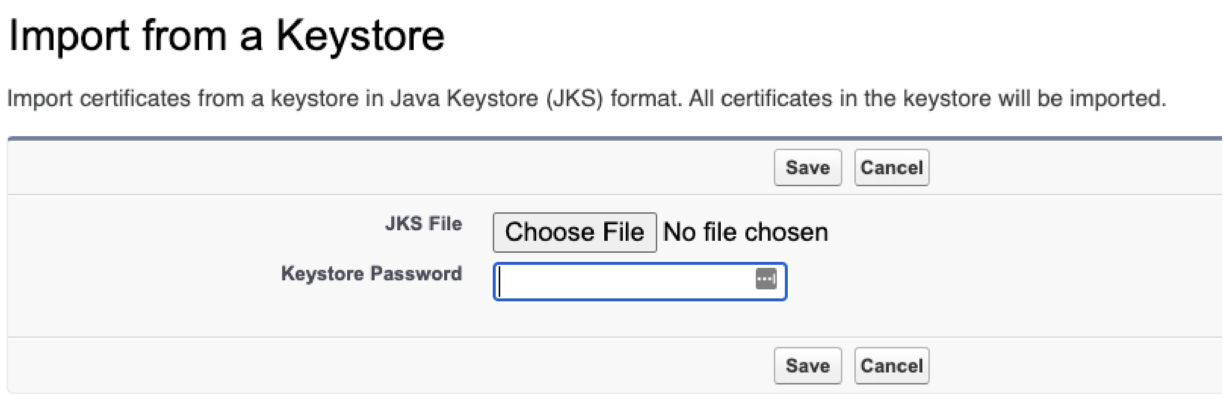
Create, update, and archive your keys based on your organization's security needs.

For increased security, specify a certificate to use as your org's API client certificate. The API client certificate is used by workflow outbound messages, the AJAX proxy, some P
callouts.

Action	Label ↑	Type	Active	Key Size	Expiration Date	Created Date
Edit | Del	jwt1234cert	Self-Signed	✓	2048	10/5/2022	10/5/2021, 1:42 PM

You've created 1 non-expired certificate out of a limit of 50.

Click on import from keystore

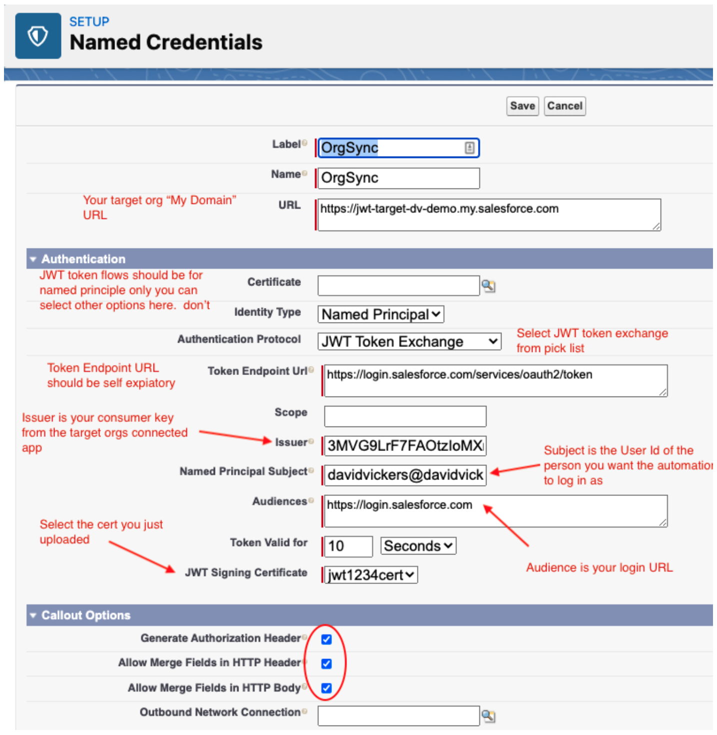

Import from a Keystore

Import certificates from a keystore in Java Keystore (JKS) format. All certificates in the keystore will be imported.

Choose file and select the JKS file you created earlier, I hope you remembered the password :-)
Click Save. If you get and error " data not available" then see the gotchas at the very beginning of this quip. Follow those steps there and try again. It will work

Your JKS file will be uploaded with the label being the Alias you set in the earlier step. Again an JKS file with the default alias of 1 will not be uploaded so make sure you followed the last keytool step in the previous instructions.

Ok now we go to Setup → Security → Named Credentials
Set the nabed credential as shown. your URL , named principal etc will of course be different


SETUP

Named Credentials

Label	OrgSync
Name	OrgSync
URL	https://jwt-target-dv-demo.my.salesforce.com

Your target org "My Domain" URL

Authentication

JWT token flows should be for named principle only you can select other options here. don't

Certificate	
Identity Type	Named Principal
Authentication Protocol	JWT Token Exchange
Token Endpoint Url	https://login.salesforce.com/services/oauth2/token
Scope	
Issuer	3MVG9LrF7FAOtzIoMX
Named Principal Subject	davidvickers@davidvick
Audiences	https://login.salesforce.com
Token Valid for	10 Seconds
JWT Signing Certificate	jwt1234cert

Select JWT token exchange from pick list

Token Endpoint URL should be self expiatory

Issuer is your consumer key from the target orgs connected app

Subject is the User Id of the person you want the automatio to log in as

Select the cert you just uploaded

Audience is your login URL

Callout Options

Generate Authorization Header	✓
Allow Merge Fields in HTTP Header	✓
Allow Merge Fields in HTTP Body	✓
Outbound Network Connection	

Save this and you should be ready to go

Now let's see if your set up has worked

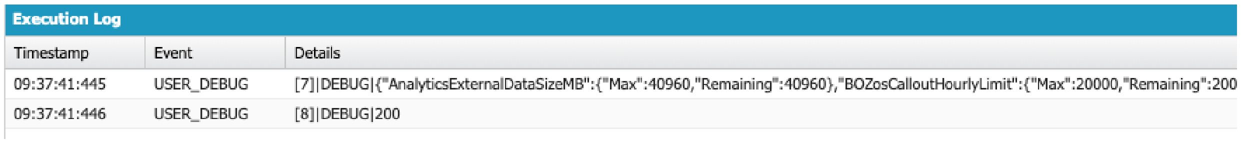
Open an developer console
Debug → Open Execute Anonymous Window (CTRL+E)
past in the following

```
string service_limits = '/services/data/v52.0/limits';
httpRequest req = new httpRequest();
req.setEndpoint('callout:OrgSync'+service_limits);
req.setMethod('GET');
http h = new Http();
httpResponse res = h.send(req);
System.debug(res.getBody());
system.debug(res.getStatusCode());
```

NOTE ('callout:OrgSync') Orgsync was what I named my named credential , if you named yours differently then you need the name of your org credential here.

In this code I am adding the limits api endpoint to create the full endpoint URL you can use any endpoint available in your target org.

If all the steps have worked you should get two debug lines the first with your limits status the second with 200


Execution Log

Timestamp	Event	Details
09:37:41:445	USER_DEBUG	[7]|DEBUG|{"AnalyticsExternalDataSizeMB":{"Max":40960,"Remaining":40960},"BOZosCalloutHourlyLimit":{"Max":20000,"Remaining":200
09:37:41:446	USER_DEBUG	[8]|DEBUG|200

That’s it you have now connected two salesforce orgs with a JWT token flow.

OK SO WHAT IS YOU HAVE AN EXTERNAL APP AY PYTHON BASED?

No Problem you target org needs a connected app set up Just as we did above. Then Try the code below!
The Code simply needs the Private Key and your consumer Key and then it constructs a valid JWT token
your Json payload to construct the cert should contain
iss = Consumer key from your connected app
exp = expiration time in unix time code format
aud = https://login.salesforce.com. or https://test.salesforce.com (later for sandbox login)
sub = The user name that the token will be used to log in as

```
import jwt
import time
import requests
import json
import os
key_file='key_file.key'

from dotenv import load_dotenv
load_dotenv()

myurl= os.getenv('url')
issuer = os.getenv('iss')
user = os.getenv('user')

with open(key_file) as fd:
  private_key = fd.read()

payload = {
      'iss': issuer,
      'exp': int(time.time()) + 300,
      'aud': myurl,
      'sub': user
    }

encoded = jwt.encode(payload, private_key, algorithm='RS256')
print('JWT:', encoded )

r = requests.post(myurl + '/services/oauth2/token', data = {
    'grant_type': 'urn:ietf:params:oauth:grant-type:jwt-bearer',
    'assertion': encoded,
  })

print('Status:', r.status_code)
access_token = r.json().get("access_token")
instance_url = r.json().get("instance_url")
print("Access Token:", access_token)
print("Instance URL", instance_url)
```

.env file

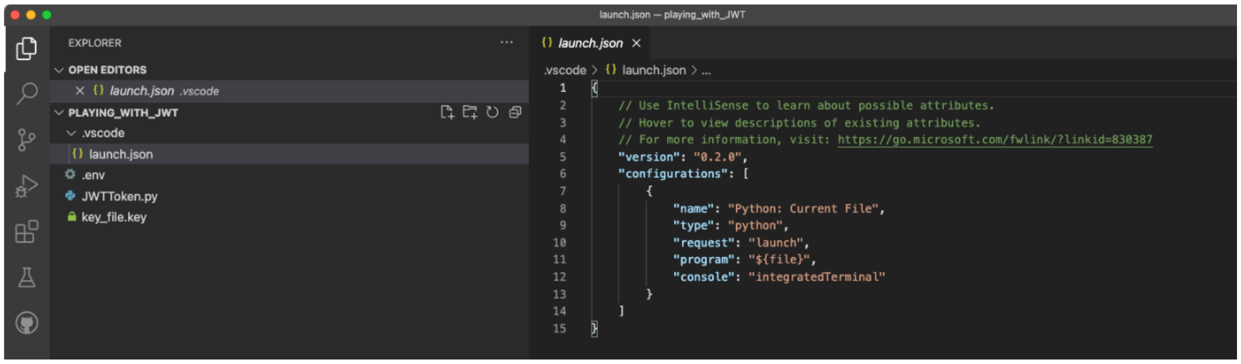
```
iss=YOUR CONSUMER KEY
user=YourUserNAME
url=https://login.salesforce.com
```

and I put the Private key in the same VS code folder as the python project
/Users/david.vickers/Python_Projects/playing_with_JWT/key_file.key

That way

```
with open(key_file) as fd:
  private_key = fd.read()
```

can read the keyfile directly no need for paths!

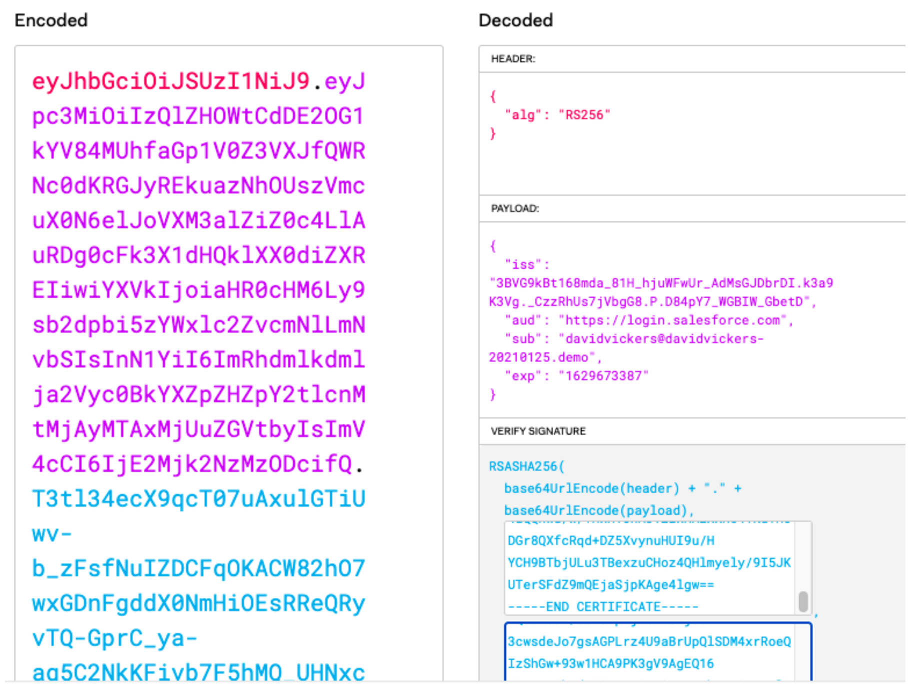


```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Current File",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal"
        }
    ]
}
```

Note that we can copy paste our JWT token out from the terminal as

```
print('JWT:', encoded )
```

will cause echo out your token that you could use for testing in Postman but you can also manually create a JWT token here https://jwt.io/



Encoded

```
eyJhbGciOiJSUzI1NiJ9.eyJ
pc3MiOiIzQlZHOWtCdDE2OG1
kYV84MUhfaGp1V0Z3VXJfQWR
Nc0dKRGJyREkuazNhOUszVmc
uX0N6elJoVXM3alZiZ0c4LlA
uRDg0cFk3X1dHQklXX0diZXR
EIiwiYXVkIjoiaHR0cHM6Ly9
sb2dpbi5zYWxlc2ZvcmNlLmN
vbSIsInN1YiI6ImRhdmlkdml
ja2Vyc0BkYXZpZHZpY2tlcnM
tMjAyMTAxMjUuZGVtbyIsImV
4cCI6IjE2Mjk2NzMzODcifQ.
T3tl34ecX9qcT07uAxulGTiU
wv-
b_zFsfNuIZDCFqOKACW82h07
wxGDnFgddX0NmHiOEsRReQRy
vTQ-GprC_ya-
aq5C2NkKFivb7F5hMO_UHNxc
```

Decoded

HEADER:

```
{
  "alg": "RS256"
}
```

PAYLOAD:

```
{
  "iss":
"3BVG9kBt168mda_81H_hjuWFwUr_AdMsGJDbrDI.k3a9
K3Vg._CzzRhUs7jVbgG8.P.D84pY7_WGBIW_GbetD",
  "aud": "https://login.salesforce.com",
  "sub": "davidvickers@davidvickers-
20210125.demo",
  "exp": "1629673387"
}
```

VERIFY SIGNATURE

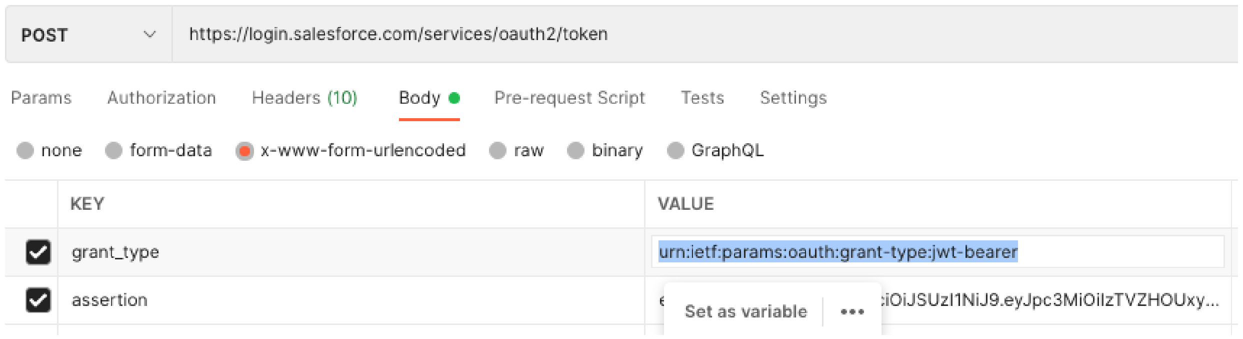
```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  DGr8QXfcRqd+DZ5XvynuHUI9u/H
  YCH9BTbjULu3TBexzuCHoz4QHlmyely/9I5JK
  UTerSFdZ9mQEjaSjpKAge4lgw==
  -----END CERTIFICATE-----
,
  3cwsdeJo7gsAGPLrz4U9aBrUpQlSDM4xRoeQ
  IzShGw+93w1HCA9PK3gV9AgEQ16
```

your exp value must be a valid unix time stamp use https://www.unixtimestamp.com/ to get a current value or the log in will fail

A valid JWT Token can then be used in postman

The grant type is urn:ietf:params:oauth:grant-type:jwt-bearer
The assertion is your valid JWT token

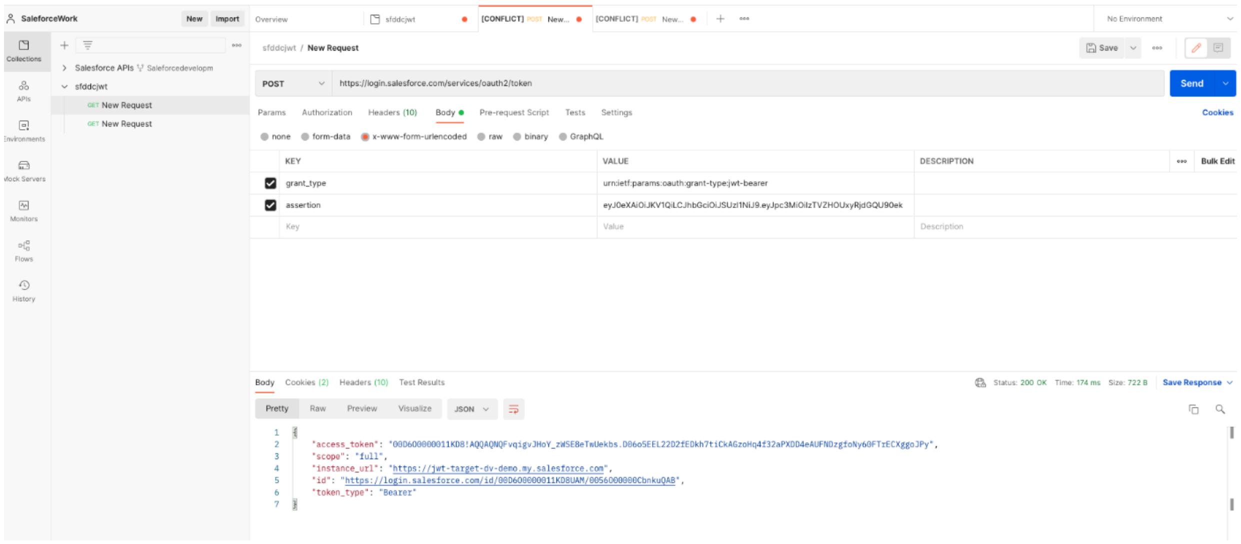
The set up will be like this



POST https://login.salesforce.com/services/oauth2/token

	KEY	VALUE
✓	grant_type	urn:ietf:params:oauth:grant-type:jwt-bearer
✓	assertion	e… :iOiJSUzI1NiJ9.eyJpc3MiOiIzTVZHOUxy...

When you send it you should get a token back



POST https://login.salesforce.com/services/oauth2/token

Body: [✓ x-www-form-urlencoded]

	KEY	VALUE	DESCRIPTION
✓	grant_type	urn:ietf:params:oauth:grant-type:jwt-bearer	
✓	assertion	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiIzTVZHOUxyRjdGQU90ek	

Status: 200 OK Time: 174 ms Size: 722 B

```
{
    "access_token": "00D600000011KD8!AQQAQNQFvqigvJHoY_zWSE8eTwUekbs.D06o5EEL22D2fEDkh7tiCkAGzoHq4f32aPXDD4eAUFNDzgfoNy60FTrECXggoJPy",
    "scope": "full",
    "instance_url": "https://jwt-target-dv-demo.my.salesforce.com",
    "id": "https://login.salesforce.com/id/00D600000011KD8UAM/0056000000CbnkuQAB",
    "token_type": "Bearer"
}
```