Installation Guide



Explore and Checkout (DC LWC)

**Table Of Contents**

[**Installing the Org DX DataPack**](#_heading=h.trx5amxqi5ok) **4**

[**Installing the Vlocity DC VIP MultiPack file containing the VIP for the checkout process.**](#_heading=h.gqocq3g74cz4) **7**

[**Launching a Digital Commerce Sample LWC App**](#_heading=h.ryz04nkbritz) **7**

[**Downloading and Using the NDS (Newport Design System) file**](#_heading=h.k7agjtarytkg) **7**

**Setting up a Digital Commerce Server SDK Reference Implementation 8**

**How to use Server SDK in LWC 17**

#### **Whitelisting URL in Secure Server App 19**

**Setting up Cross-Origin Resource Sharing (CORS) in Salesforce 20**

**Create CSP Trusted Sites to Access Third-Party APIs 20**

# 

# 

# 

# 

# 

# 

# 

Access to the Vlocity Process Library requires acceptance of the disclosure below:

**RESTRICTED CONTENT: AUTHORIZED USE ONLY**

***The content on the following pages is intended to be viewed only by Vlocity employees, contractors who have been authorized by Vlocity or customers who have been authorized by Vlocity. The content should be considered the sensitive confidential information of Vlocity and is protected under the nondisclosure agreement entered into between your party and Vlocity. Accordingly, further distribution is prohibited without prior written consent from Vlocity.***

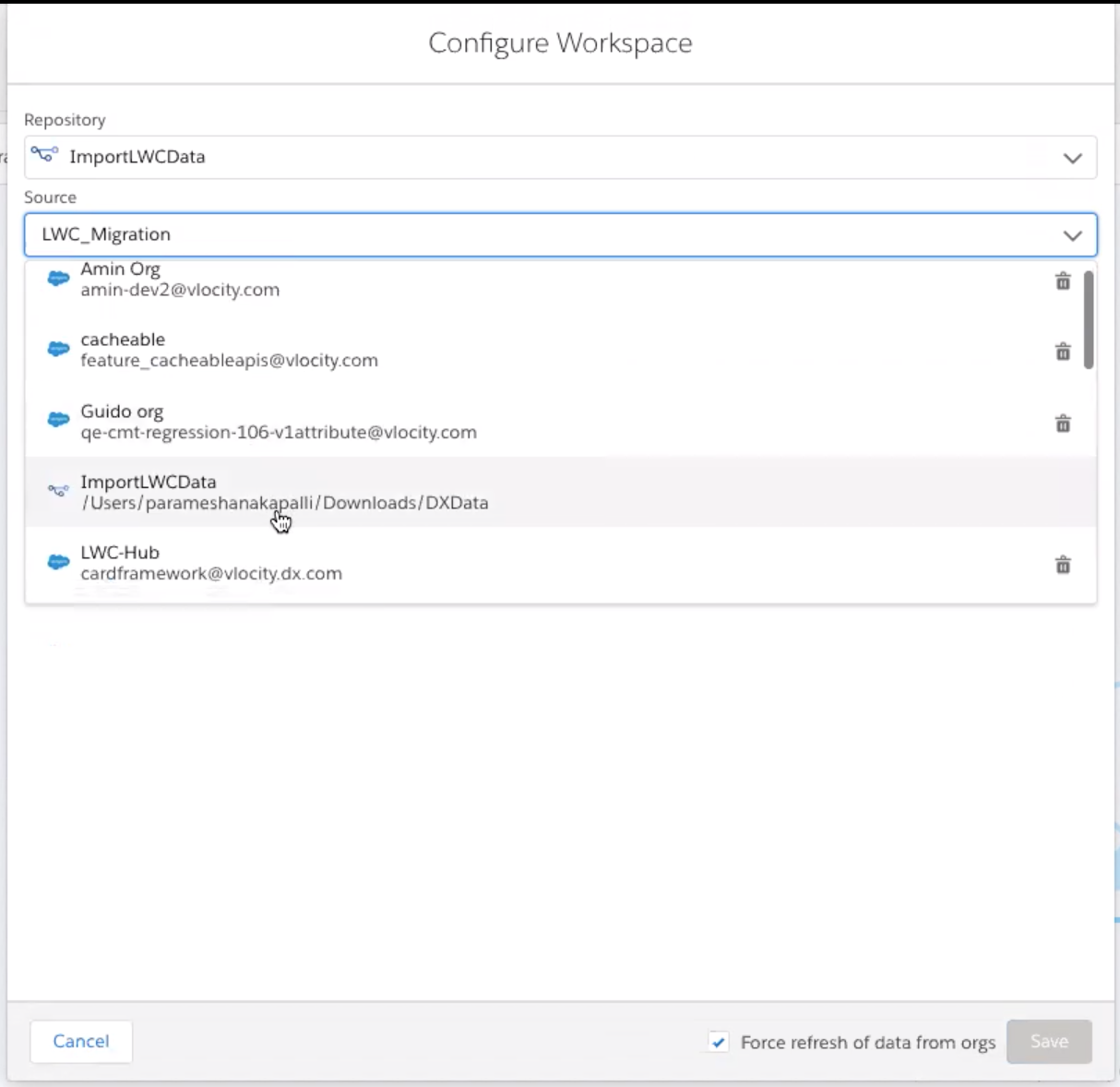
***I Accept Vlocity’s Terms and Conditions*** [***https://vlocity.com/terms***](https://vlocity.com/terms)

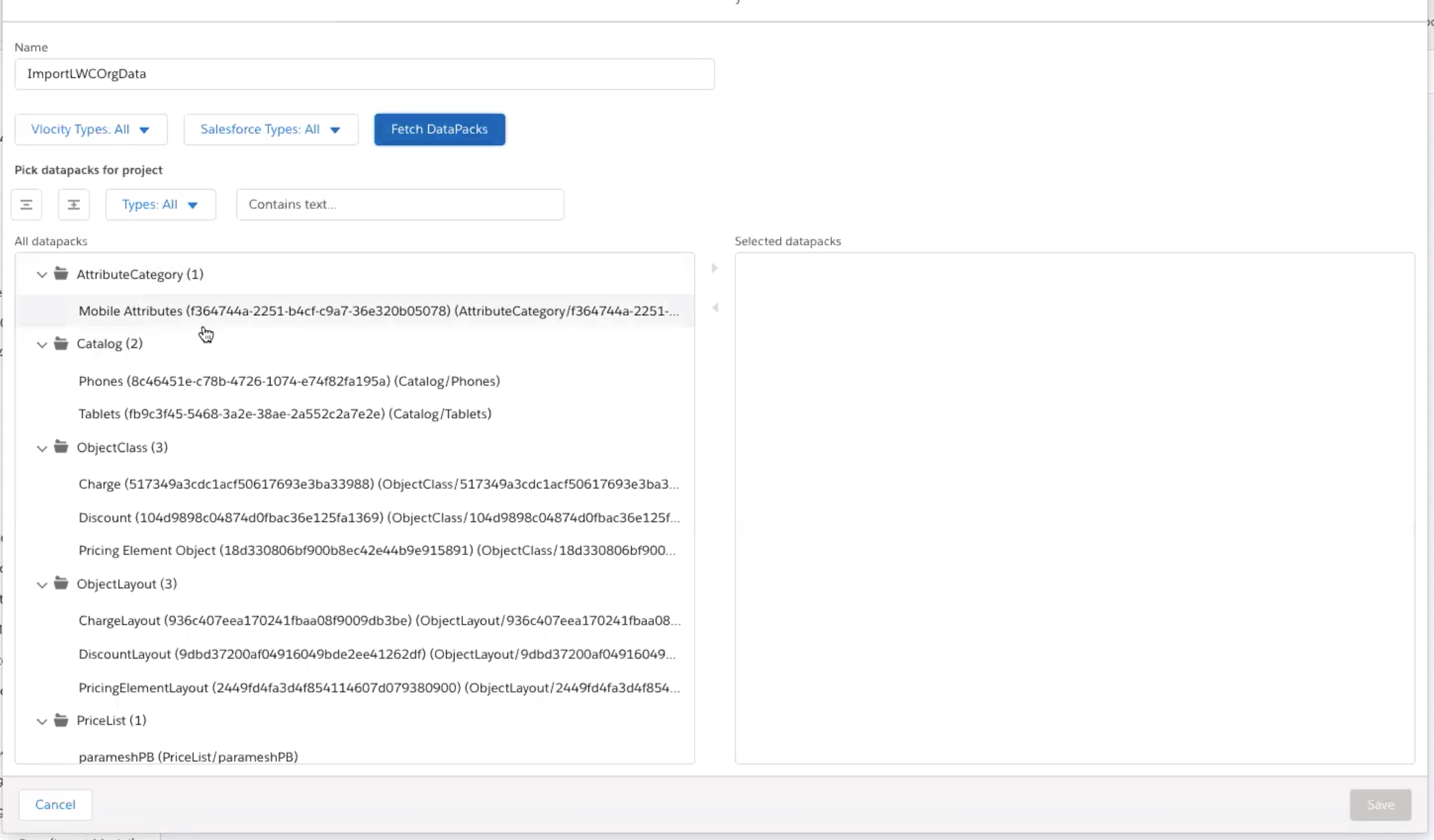
# 

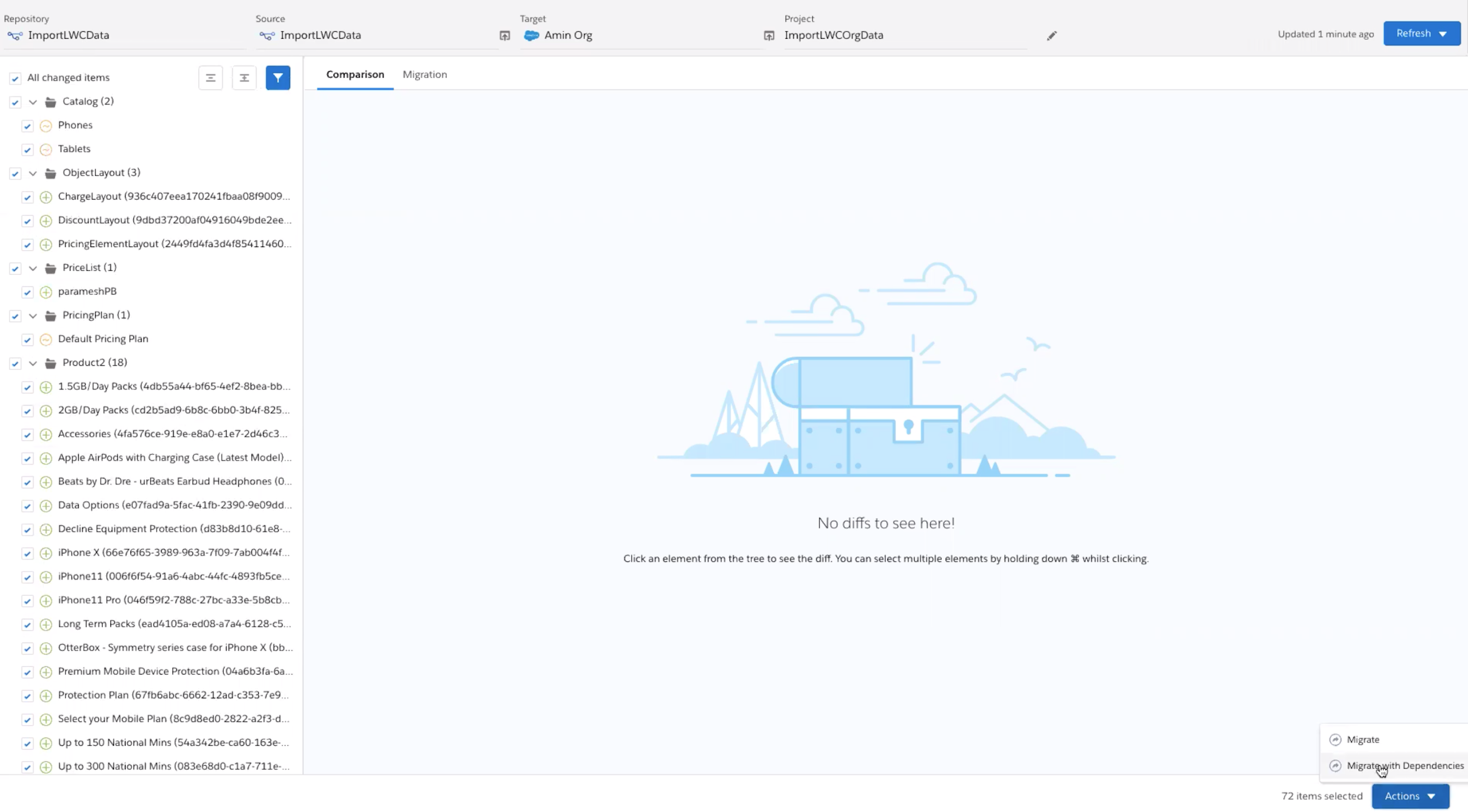
# **Installing the** **Org DX DataPack**

To run the sample application, you use the Vlocity DX desktop application to migrate the Vlocity DataPack from the Vlocity Process Library to your org.  Follow the steps below to download and run Vlocity DX.

1. Install the latest DX tool from: [https://docs.vlocity.com/en/Vlocity-DX-Desktop-Application.htm](https://docs.vlocity.com/en/Vlocity-DX-Desktop-Application.html)
2. Download the DX org datapack and unzip it.
3. Open the DX tool and select your unzipped DX org DataPack folder as the repository.
4. Select the source as your unzipped DX org DataPack folder.
5. Specify your salesforce org as the target.
6. Click on a new project and provide a name (such as "ImportDCOrgData").
7. Select **VlocityTypes** as **all** and **Salesforce Types** as **all**.
8. Click **Fetch DataPacks** and then click **Save**.
9. Click **Migrate With Dependencies**.







# **Launching a Digital Commerce Sample LWC App Without Proxy SDK**

1. Create a new LWC component say dcSampleAppTest, and include dcSampleApp component in the HTML.
2. Make sure that you are passing checkoutPaymentUrl, checkoutApiUrl and authConfiguration details via attributes.
3. From Setup, enter App Builder in the Quick Find box, then select **Lightning App Builder**.
4. Click **New**.
5. Select **App Page**, and then click **Next**.
6. Name your Lightning page Digital Commerce
7. Select the **One Region** template, and click **Finish.**
8. Drag the dcSampleAppTest into the region.
9. Click the **Activation** button and activate it.
10. Now to test your app, click **Back** to return to Setup If you’re still in the App Builder.
11. From the App Launcher (App Launcher icon), find and select **Digital Commerce**.
12. Click **Digital Commerce** from the app navigation bar to open the app.

# **Launching a Digital Commerce Sample LWC App With Proxy SDK**

1. Create a new LWC component say dcSampleAppTest, and include dcSampleApp component in the HTML.
2. Make sure that you are passing checkoutPaymentUrl, checkoutApiUrl, authConfiguration details and enableProxySdk = true via attributes.
3. From Setup, enter App Builder in the Quick Find box, then select **Lightning App Builder**.
4. Click **New**.
5. Select **App Page**, and then click **Next**.
6. Name your Lightning page Digital Commerce
7. Select the **One Region** template, and click **Finish.**
8. Drag the dcSampleAppTest into the region.
9. Click the **Activation** button and activate it.
10. Now to test your app, click **Back** to return to Setup If you’re still in the App Builder.
11. From the App Launcher (App Launcher icon), find and select **Digital Commerce**.
12. Click **Digital Commerce** from the app navigation bar to open the app.

# **Downloading and Using the NDS (Newport Design System) file**

1. Clone the project with the following command: git clone https://github.com/vlocityinc/newport-design-system.git
2. Change it to the **newport-design-system** folder using **cd** **newport-design-system**
3. Switch to the right branch for your version of the package, for example, git checkout v109
4. Run **npm install**
5. Run **npm start** to launch storybook
6. Goto UI/components/digitalcommerce
7. Modify styles as per requirement.
8. Compile your CSS changes using **npm run build && npm run dist**
9. Go to dist folder and upload **newport-design-system.zip** file into your org static resources.
10. Provide a name for the static resource file, such as **newport\_dc.**
11. Select **Cache Control** as public and click **Save**.
12. Open your uploaded static resource newport\_dc file again and do right click and click on **copy link address** option.
13. Go to Setup and search for **Custom Settings**.
14. Search for **UISettings** and click **Manage**.
15. Click the **New** button and create a new setting specify a name such as **newportZipUrl**
16. Paste the copied URL into the **value** input text box.
17. Refresh your application to see the updated styles.

**Setting up of Digital Commerce Server SDK Reference Implementation**

Server SDK Reference Implementation app is a node application that could be deployed to any of the hosting servers for eg. Heroku, AWS, etc

### **Step 1: Download Server SDK Reference App** Download Server SDK Reference App from the VPL.

### **Step 2: SF Org Setup**

In Server SDK we are using OAuth 2.0 JSON Web Token (JWT) bearer flow to authenticate a server. Therefore, we re require to -

* Generate a public/private key pair
* Create a Connected App

**Generating public/private key pair**

Generate a public/private keypair using openssl and fill in the required info when you generate the certificate.

openssl req -newkey rsa:2048 -nodes -keyout private\_key.pem -x509 -days 365 -out certificate.pem

openssl x509 -outform der -in certificate.pem -out public\_key.der

openssl x509 -in certificate.pem -pubkey > public\_key.pem  
See Doc: <https://github.com/lekkimworld/salesforce-jwt-generator>

#### **Create connected APP & upload CER file (ie certificate) to SFDC**

In Salesforce create a Connected App through the App Manager in Setup and upload the public key (public\_key.cer from the above steps) to the app. Be sure to select the offline\_access scope as well as other required scopes. Save the Connected App and make a note of the **consumer key (issuer\_id).**See Doc: <https://help.salesforce.com/articleView?id=connected_app_create.htm&type=5>

### **Step 3: Auth Config**

Server SDK app has an authentication layer which is responsible for validating a user identity against an authentication server like Firebase, Cognito etc. A reference implementation using Firebase as an authentication strategy has been used in the app by default. However, you could easily swap firebase for any other third party authentication system by creating a custom authentication strategy.

* Create a folder inside *vop\_server\_sdk/secure-server-reference-app/authentication/strategies* directory and inside it create a file and Implement IAuthenticationStrategy interface. The implementing class should define the properties with the corresponding method of the third party authentication system. For example, the implementation of signIn method should have the signIn logic of your third party authentication system that you want to use.
* Once you have created your custom authentication strategy, open *AuthenticationStrategy.ts* file and import your customise strategy and initialise it inside the constructor.

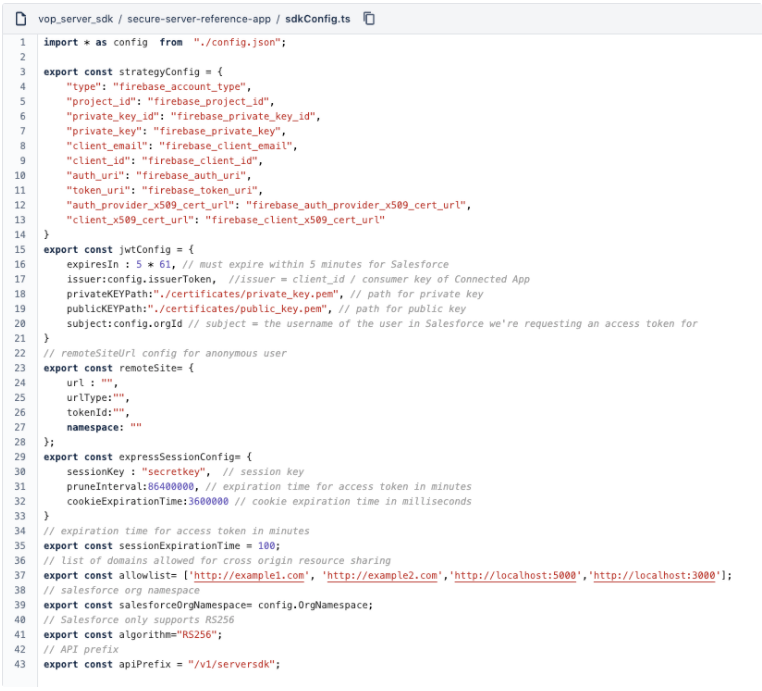
### **Step 4: Setting up** config.json **file**

find config.json file in vop\_server\_sdk/secure-server-reference-app/config.json  


* Enter the **consumer key (issuer\_id)** which you noted down in step 2.
* **org Id -** Enter the **username** of the salesforce user.
* **org namespace** - enter the SF org namespace.

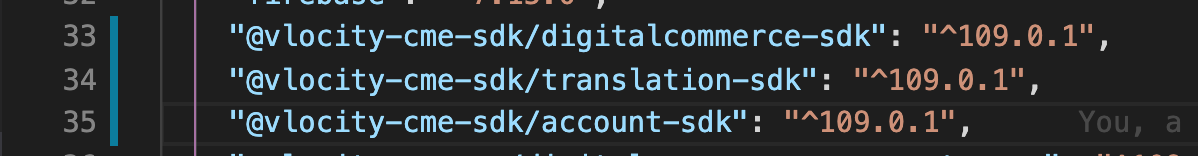
### 

### **Step 5: Setting up** sdkConfig.ts **file**

find sdkConfig.ts file in vop\_server\_sdk/secure-server-reference-app/sdkConfig.ts  


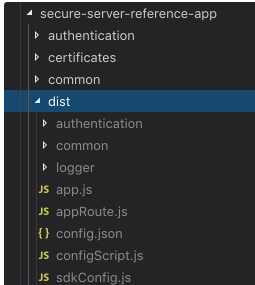
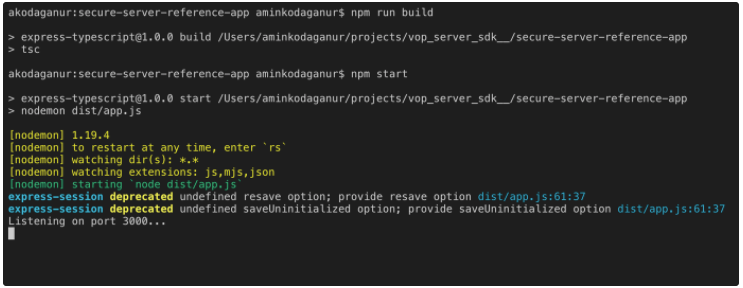
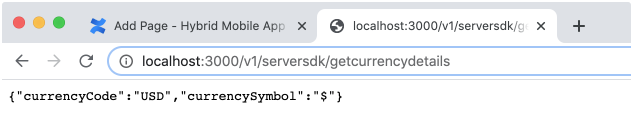
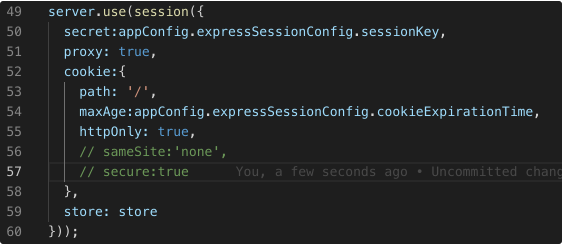
* **strategyConfig**: If you're using default firebase as authentication server then just set a strategyConfig object.
* **jwtConfig:** Paste private\_key.pem & public\_key.pem files generated from step2 in vop\_server\_sdk/secure-server-reference-app/certificates folder for JWT sign
* **remoteSite:** for anonymous users, url will be anonymous URL of org & namespace will be org namespace.
* **allowlist:** when you're requesting from cors domain then you need to whitelist that domain here.

### **Step 6: Setting up** package.json



* In vop\_server\_sdk/secure-server-reference-app/package.json file under dependencies object please update SDKs with 109 version  
  **Note:** please make sure you add a .npmrc file. The .npmrc file is required to authenticate with the Vlocity private npm registry. For public access please use the default access token.

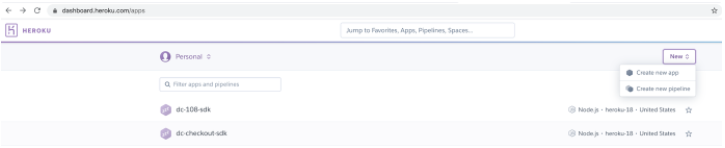
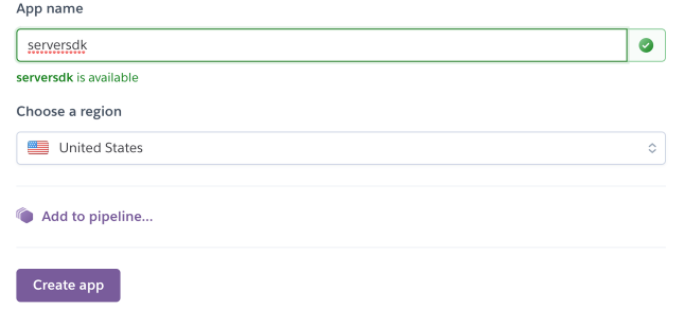
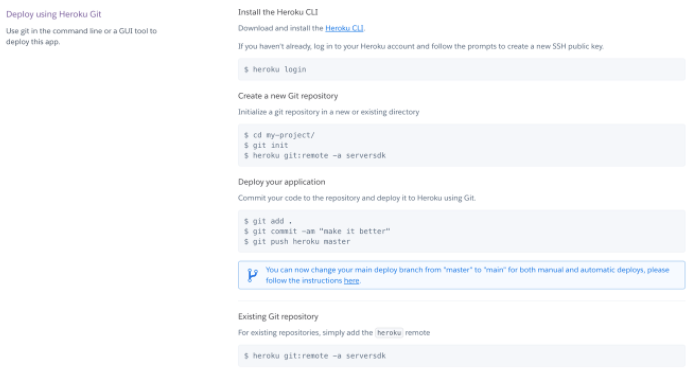
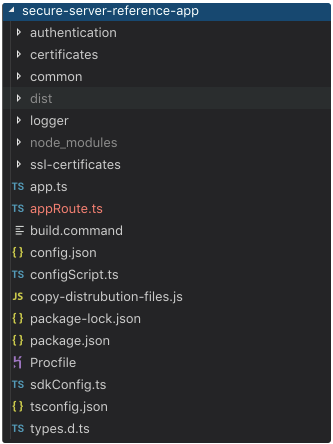
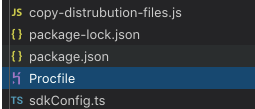
### **Step 7: Running in local machine (Optional)**

* Open root directory(vop\_server\_sdk/secure-server-reference-app) in the terminal.
* run $ npm install will install all dependency.
* run command below commands  
    
   $ npm run build - will take few seconds to compile & build typescript files  
  Once this is successful it will generate /dist folder in the root directory.  
    
  Now the $ npm start will run the server.  
    
    
    
  Your terminal logs should look like below  
    
  By default server will start listening to port no 3000 so now base API URL will be  
  http://localhost:3000/v1/serversdk  
    
   Just to test if the api is working fine you can hit http://localhost:3000/v1/serversdk/getcurrencydetails in the browser it will fetch default currency details.  
    
  **Note:** make sure you comment below lines in vop\_server\_sdk/secure-server-reference-app/app.ts these settings required only when you host in third party server like heroku  
  

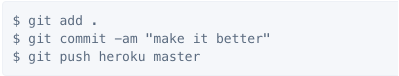
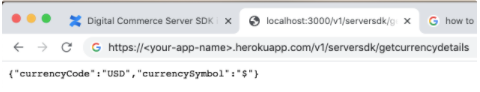
### 

### **Step 8: Deploying in heroku**

You can deploy/host this ServerSDK application in any node supported server. one of famous server is heroku below are steps to deploy/host in same

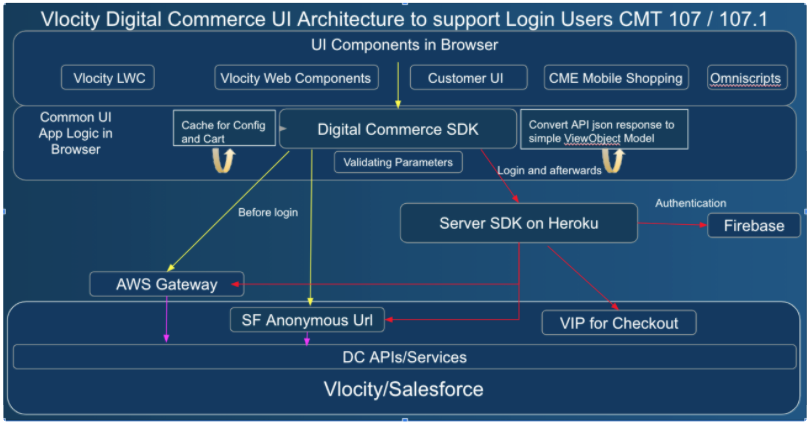
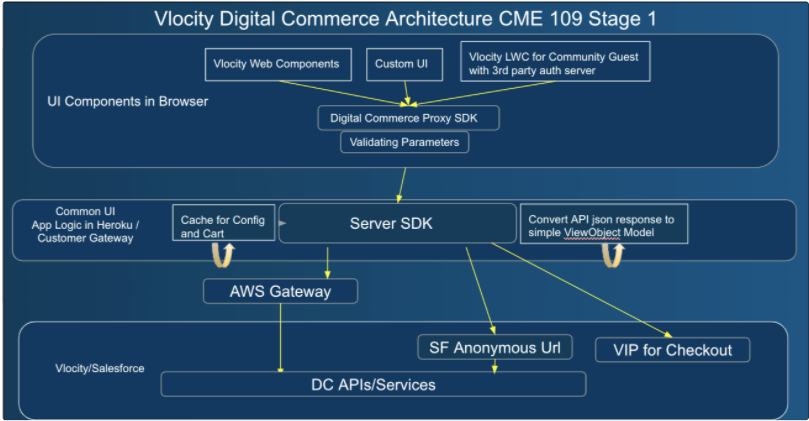
* Login to your Heroku account  
  <https://id.heroku.com/login>
* Click New button & select Create a new App from Dashboard  
  
* Enter app name & click create app button  
  
* Navigate to deploy tab  
  
* Follow Deploy using heroku git doc  
  
* After following the above step you should have a folder where you have cloned/setup with git heroku origin.  
  now we need to copy our serverSDK into heroku cloned/setup folder  
  Copy entire files under vop\_server\_sdk/secure-server-reference-app  
    
  Note: no need to copy dist & node\_modules folder
* create a new file with Procfile in root of heroku cloned/setup folder. It should look like below  
  

Paste below contain into Procfile file  
web: npm run start-prod-server  
this will run our serverSDK in heroku after code push.

* Now open heroku cloned/setup folder in terminal, commit & push the files to heroku.  
  
* Once you push code Heroku will automatically detect as NodeJS app it will create build & start serving. Your log should look like below once you push  
    
    
  Notice it will log base API URL at end in above case it’s https://server-sdk-109-pref.herokuapp.com/
* Once you pushed & deployed successfully your base URL will be  
   https://<your-app-name>.herokuapp.com/v1/serversdk  
  You can also find it in Heroku dashboard > Settings > Domains section  
  **Note:** /v1/serversdk is a default API prefix you can change it vop\_server\_sdk/secure-server-reference-app/sdkConfig.ts
* To verify if heroku setup is working fine open browser and hit below your  
   https://<your-app-name>.herokuapp.com/v1/serversdk/getcurrencydetails it will fetch default currency details  
  

## **How to use Server SDK in LWC :**

We can you use Server SDK in two ways

* **With Client side SDK:** This is same as v107 implementation where DC API will be called directly from client for guest user after signIn all DC API will be called from ServerSDK below is architecture of it  
  
* **With Proxy SDK:**In this case all DC API will be called from Server SDK. We can easily implement it by using Proxy SDK library which is very light weight and shares the same signatures as other SDK so that there will be no change required in UI code to support this. Below is architecture of it  
  

**DC** **LWC Setup:**

In your LWC component html include <c-dc-sample-app></c-dc-sample-app> component with below attribute.

<c-dc-sample-app

checkout-api-url=""

secure-server-api-request-credentials=""

checkout-payment-url=""

enable-proxy-sdk=""

auth-configuration=""

>

</c-dc-sample-app>

The attribute (checkout-api-url) has value which contains the url where server SDK is hosted.

The attribute (secure-server-api-request-credentials) has value which contains value indicating whether the user agent should send cookies to the secure server if it's hosted on another domain in the case of cross-origin requests. Possible values are:include,omit and same-origin(default value)

The attribute (checkout-payment-url) has value which contains checkout payment URL.

The attribute (enable-proxy-sdk) has a value which contains boolean value true or false.If don’t specify by default it will be false . If it set to true: Digital Commerce, Account, Translation Proxy SDK instance will be created

The attribute (auth-configuration) will be your firebase or custom auth configuration details. You can create custom auth by overriding *dcCustomAuthenticationUtil*

#### **Whitelisting URL in Secure Server App**

Secure server reference app uses cookies to persist user sessions. These cookies are not sent by default over CORS.Therefore, you need to whitelist the client URL from where you would be accessing the server. This could be configured in the property whiteList of **sdkConfig.ts** filepresent in the root directory.

**Fixing CORS issue on client side**

If the origin of your client application is different from that of the secure server you would get a CORS issue due to [Same-origin policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy) and hence resource sharing between client and server will not take place. Therefore to fix the CORS issue you need to follow two steps -

1. Whitelist the client origin URL on the server SDK reference app.
2. Make sure to include credentials: 'include' in the request header options for REST calls.

In cme 109 to add credentials: 'include' property you can override the Digital commerce SDK checkout methods to create the REST API body and attach credentials: 'include' in the options of the request header. Then using datasource SDK we can invoke the REST API call to the server.

# **Setting up CORS in salesforce**

Cross-Origin Resource Sharing (CORS) enables web browsers to request resources from origins other than their own (cross-origin). For example, using CORS, JavaScript code at https://www.example.com could request a resource from https://www.salesforce.com. To access supported Salesforce APIs, Apex REST resources, and Lightning Out from JavaScript code in a web browser, add the origin serving the code to a Salesforce CORS whitelist.

1. From Setup, enter CORS in the Quick Find box, then select CORS.
2. Select New.
3. Enter an origin URL pattern.  
   The origin URL pattern must include the HTTPS protocol (unless you’re using your localhost) and a domain name and can include a port. The wildcard character (\*) is supported and must be in front of a second-level domain name. For example, https://\*.example.com adds all subdomains of example.com to the whitelist.  
   The origin URL pattern can be an IP address. However, an IP address and a domain that resolve to the same address are not the same origin, and you must add them to the CORS whitelist as separate entries.  
   The origin URL pattern might not match the URL that appears in the address bar in your browser. Make sure that you’re whitelisting the origin in the request header.

Note - For the salesforce document on CORS follow this [link](https://help.salesforce.com/articleView?id=extend_code_cors.htm&type=5).

# **Create CSP Trusted Sites to Access Third-Party APIs**

The Lightning Component framework uses Content Security Policy ([CSP](http://www.w3.org/TR/CSP/)) to impose restrictions on content. The main objective is to help prevent cross-site scripting ([XSS](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))) and other code injection attacks. To use third-party APIs that make requests to an external (non-Salesforce) server or to use a WebSocket connection, add a CSP Trusted Site.

1. From Setup, enter CSP in the Quick Find box, then select CSP Trusted Sites.
2. Select New Trusted Sites.
3. Enter Trusted Site Name
4. Enter Trusted Site URL

You can use the wildcard character \* (asterisk) to reduce repetition. The URL must include a domain name, and can include a port. For example, https://example.com:8080. For a third-party API, the URL must begin with https://.

1. Enter Trusted Site Description

Note - For the salesforce document on creating CSP Trusted Sites follow this [link](https://help.salesforce.com/articleView?id=csp_trusted_sites.htm&type=5).

**Note**: Post Winter’21 , user need permission to access an Apex class containing an @AuraEnabled method to access in LWC’s.

**How to update profiles and permissions to access Apex with @AuraEnabled**

In setup, you can add Apex classes to a profile under “Enabled Apex Classes Access”. For a permission set, you’ll see it under Apps as “Apex Class Access”.