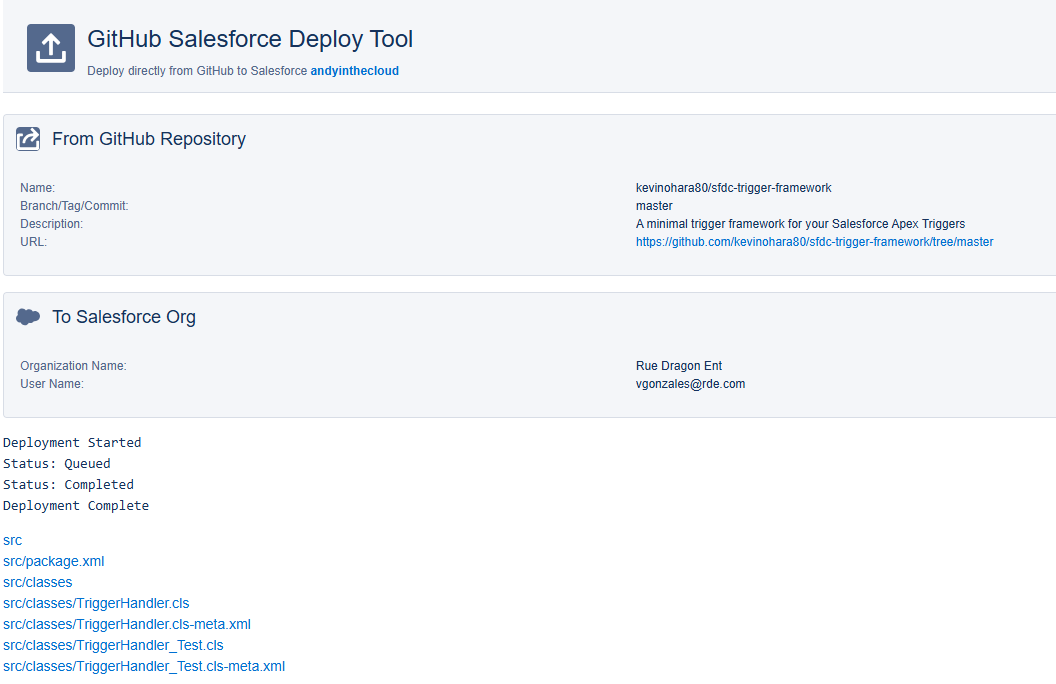
SF CodeBank Documentation

# Initial Configuration Steps

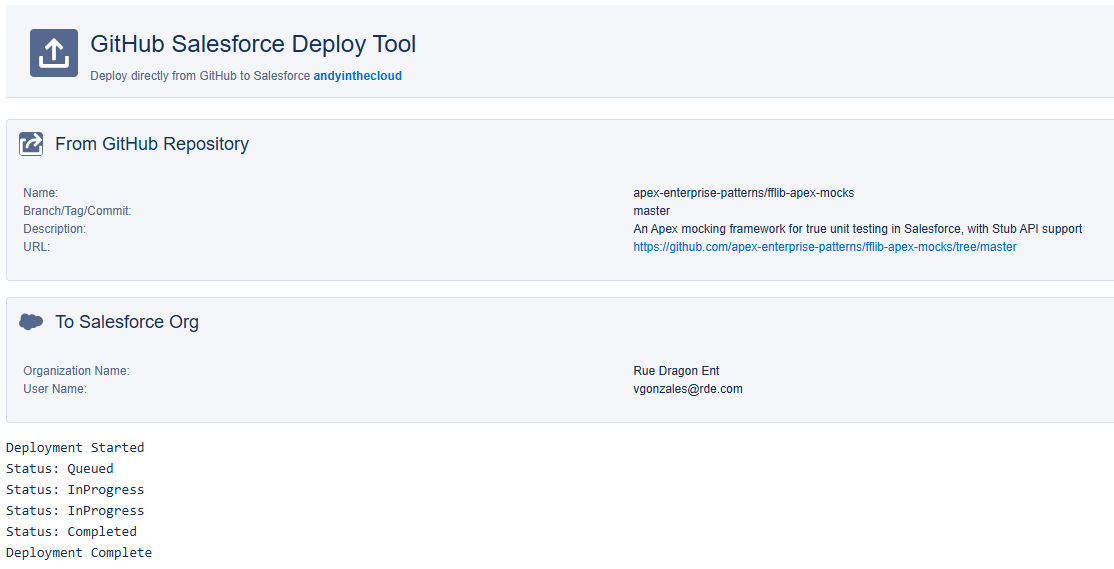
## Deploy sfdc-trigger-framework

* [sfdc-trigger-framework](https://github.com/kevinohara80/sfdc-trigger-framework)
  + Deploy both Apex Class and Test Class one at a time respectively.



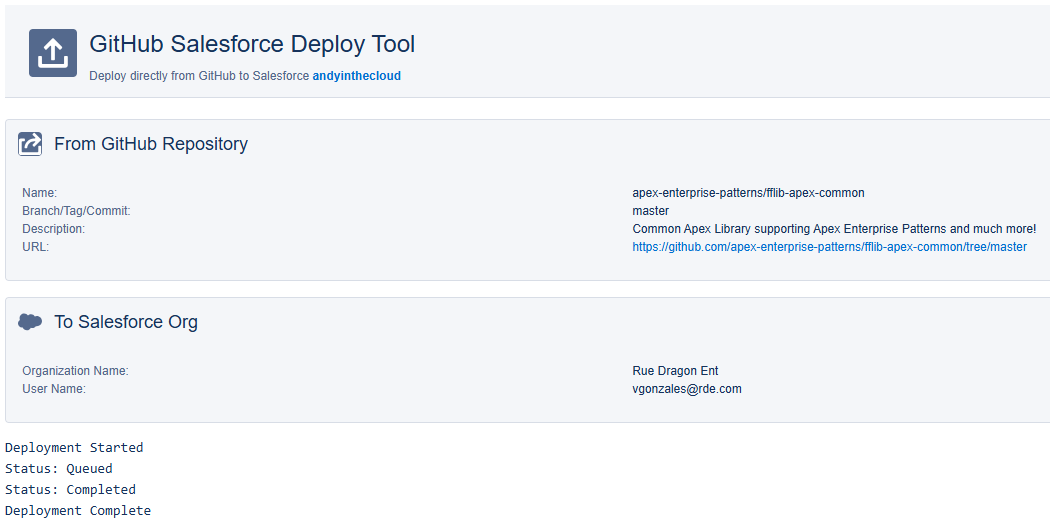
## Deploy fflib-apex-mocks

* [fflib-apex-mocks](https://github.com/apex-enterprise-patterns/fflib-apex-mocks)
  + An Apex mocking framework for true unit testing in Salesforce with Stub API support



## Deploy fflib-apex-common

* [fflib-apex-common](https://github.com/apex-enterprise-patterns/fflib-apex-common)
  + Common Apex Library supporting Apex Enterprise Patterns



# 

## Create Application Factory Class and Test Class

/\*\*

\* @description: Application Factory Class

\* @author: Veni Gonzales

\* @department: NA

\* @testClass: Application\_Test

\*/

public with sharing class **Application** {

// The constructor for this class requires you to pass a list of SObject types in the dependency order.

// So in this instance Accounts would always be inserted before your Contacts and Contacts before Cases, etc.

public static final fflib\_Application.UnitOfWorkFactory unitOfWork =

new fflib\_Application.UnitOfWorkFactory(

new List<SObjectType>{

Account.SObjectType,

Contact.SObjectType,

Opportunity.SObjectType

});

//This allows us to create a factory for instantiating service classes. You send it the interface for your service class

//and it will return the correct service layer class

//Exmaple initialization: Object objectService = Application.service.newInstance(Task\_Service\_Interface.class);

public static final fflib\_Application.ServiceFactory service =

new fflib\_Application.ServiceFactory(

new Map<Type, Type>{

});

//Update this class for every new Selector Class created

//This allows us to create a factory for instantiating selector classes. You send it an object type and it sends

//you the corresponding selectory layer class.

//Example initialization: fflib\_ISObjectSelector objectSelector = Application.selector.newInstance(objectType);

public static final fflib\_Application.SelectorFactory selector =

new fflib\_Application.SelectorFactory(

new Map<SObjectType, Type>{

//Add the selector classes here:

/\*ContentVersion.SObjectType => ContentVersionSelector.class,

ContentDocument.SObjectType => ContentDocumentSelector.class,

Opportunity.SObjectType => OpportunitySelector.class,

OpportunityLineItem.SObjectType => OpportunityProductSelector.class,

User.SObjectType => UserSelector.class\*/

});

//This allows you to create a factory for instantiating domain classes. You can send it a set of record ids and

//you'll get the corresponding domain layer.

//Example initialization: fflib\_ISObjectDomain objectDomain = Application.domain.newInstance(recordIds);

public static final fflib\_Application.DomainFactory domain =

new fflib\_Application.DomainFactory(Application.selector,

new Map<SObjectType, Type>{

});

}

/\*\*

\* @description: Test Class for Application

\* @author: Veni Gonzales

\* @department: NA

\*/

@isTest

public class **Application\_Test** {

@isTest

static void testApplication() {

test.startTest();

try {

Application app = new Application();

} catch (Exception e) {

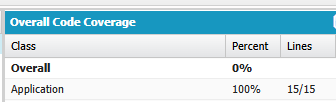
Assert.isNotNull(e);

}

test.stopTest();

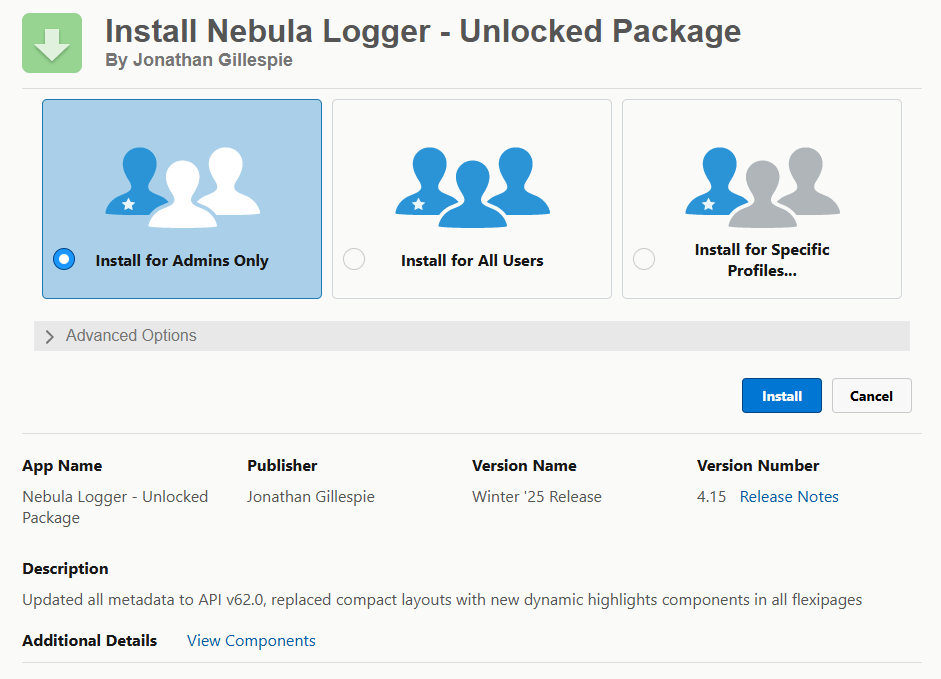
}

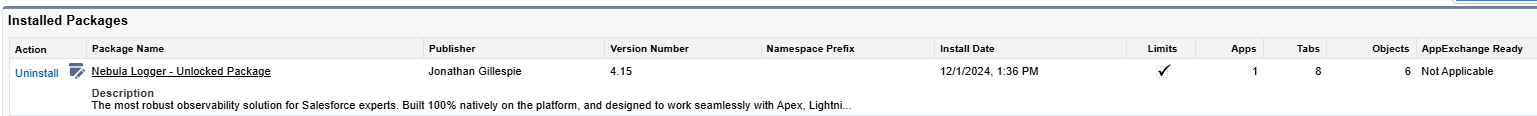
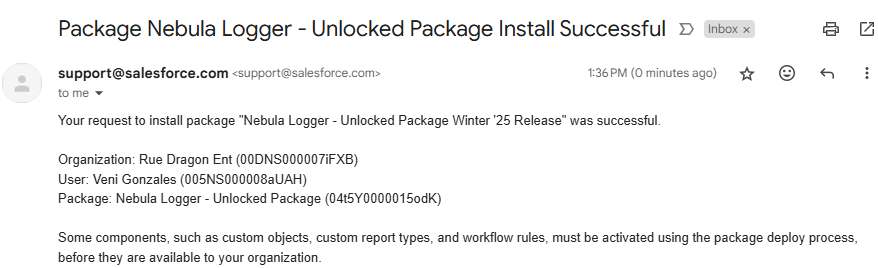
}



## Install Nebula Logger for Salesforce

* [NebulaLogger](https://github.com/jongpie/NebulaLogger) - Install the recommended Unlocked Package
* The most robust observability solution for Salesforce experts. Built 100% natively on the platform, and designed to work seamlessly with Apex, Lightning Components, Flow, OmniStudio, and integrations.

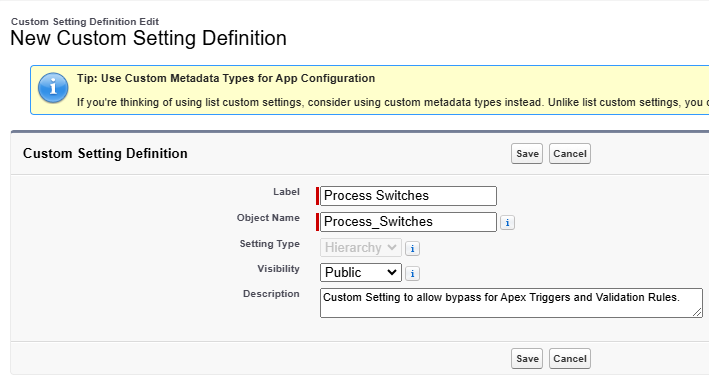




## Setup Bypass Validation Rules and Apex Triggers

**Steps:**

* Go to Setup > Quick find **Custom Settings** under Custom Code
* Click New
  + Label: Process Switches
  + Object Name: Process\_Switches
  + Visibility: Public
  + Description: Custom Setting to allow bypass for Apex Triggers and Validation Rules.



* Under the **Custom Fields** section, click the “**New**” button
* Checkbox > Next
* Field Label: Master Skip VR
* Default Value: Unchecked
* Description: Allows bypass of Validation Rules.
* Click Next > Save & New
* Checkbox > Next
* Field Label: Master Trigger Off Switch
* Default Value: Unchecked
* Description: Allows bypass on Apex Triggers.
* Next > Save



# Use Case

## Email Services

* If an email is sent by an internal active SF user
* AND the subject of the email contains Acct Ref:[XXXXXXX]
* It will look for an Account in SF with that Acct Ref number
* And attach the EmailMessage in that Account

A screenshot of a computer

Description automatically generated

* If the Account Record Type is “Private”
* The sender will receive a bounce back notification saying the email was not attached and cannot be attached to a Private Account

A screenshot of a computer

Description automatically generated

**Steps to Build:**

* Creates a new custom Text field called “Acct Ref” in the Account object
  + Length: 7
* Create Record Types called Private and Regular
* Add Record Type in the Account Layout
* Create an Apex Class called **AccountEmailHandler**
* In Email Services under Custom Code > click **New Email Service**
  + Enter an Email Service Name: Any name you want to call the Email Service
  + Apex Class: The Apex Class to handle the Messages: **AccountEmailHandler**
  + Accept Email From: Email addresses or domain of allowed email addresses

A screenshot of a computer screen

Description automatically generated

* Create the Apex Class: **AccountEmailHandler** and Test Class: **AccountEmailHandler\_Test**
* Create a **RecordTypeSelector** Class
* Create a **ConstantsGlobal** Class
* Create a **UserSelector** Class
* Create an **AccountSelector** Class